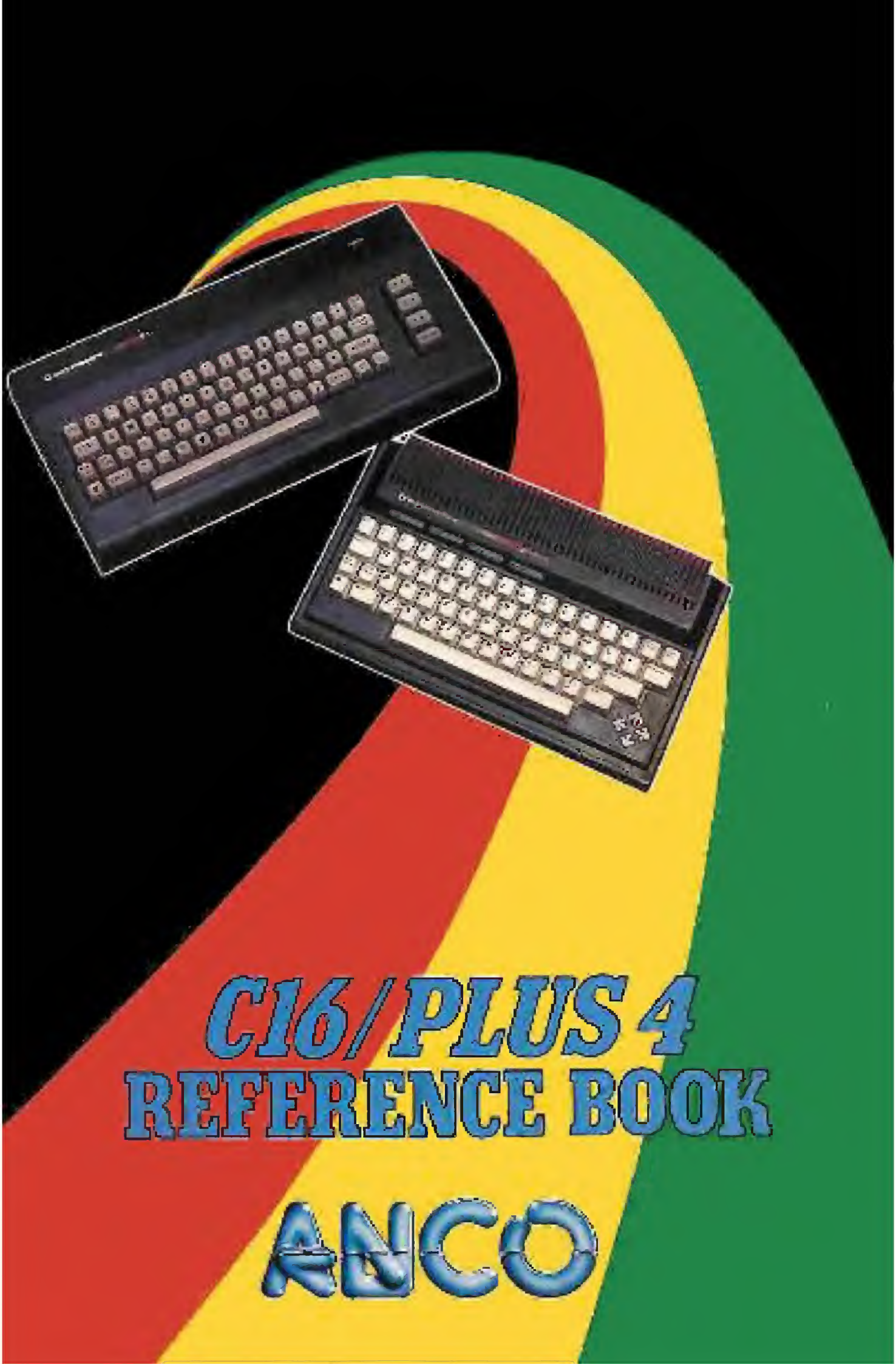

C16/PLUS 4 REFERENCE BOOK

ANCO

PREFACE

There has been a growing demand for a book that will give detailed information required to make effective use of the PLUS 4 and the C-16. This book originally published in Germany has enjoyed a huge success and excellent reviews. ANCO took an early decision to give these computers our whole hearted support. Towards this objective, we decided to get this book translated and publish it. We hope that you will make good use of mine of information available in this book.

ANCO

August 1986.

C16 AND PLUS 4 REFERENCE BOOK

CONTENTS

1. INTRODUCTION

In 1984 Commodore introduced two computer models the C-16 and the Plus 4, which inspite of excellent performance remain in the shadow of the bestseller C-64. In this book, we would like to show you that these models are grossly underestimated, because they offer among other things an extremely efficient BASIC (much better than that of the C-64), excellent graphic possibilities (320 x 200 points in 121 colours), expansion facilities and an ever increasing software base at very reasonable prices.

The main points in this book are subjects which are insufficiently or not at all dealt with in the manual supplied with the computers. It is, therefore, for all those people who want to broaden their knowledge of BASIC and by using the machine code want to make use of all the possibilities of the C-16, specially with regard to graphics and sound.

To make things easier, I will only talk about the C-16 in this book since the Plus/4 is identical (keyboard, RAM) apart from small differences. Should differences occur, I will point them out to you.

After reading this book you will know your computer better and will be able to programme more effectively. I have endeavoured to make the subjects more interesting with numerous examples since, as everybody knows, one picture says more than a thousand words.

FRITZ SCHAFFER 1986

2. GRAPHICS

One of the most impressive characteristics of all home computers are without doubt the graphics which can be created with these machines. This is one of the C-16's strongest points, because which other home computer offers a resolution of 320 x 200 points in 121 colours supported by a very efficient BASIC command set. The only handicap of the C-16 without memory expansion when programming graphics is the limited work store of 16K RAM of which 12K is available for BASIC. Very detailed graphics in many colours need a lot of space in any computer. When working in the high resolution graphics mode, the C-16 has to reserve 10K for the graphics memory which only leaves 2K RAM for the actual BASIC programme. If you want to write larger programmes in high resolution graphics (and don't have a Plus/4 with 64K built in RAM), you should consider buying a 16 K or 64 K RAM expansion. In the following chapters, I want to show you that you can also programme excellent graphics with the C-16 with the normal memory of 16K RAM. After all, almost all commercial game programmes are based on the basic version and amongst them are surely a few which appear to have a large memory but in fact don't.

2.1 STANDARD-GRAPHICS

The easiest way to create graphics on the C-16 is to use the existing character set which can be obtained from the keyboard by using the COMMODORE- and the SHIFT key. The graphics do not use any additional store space and are therefore mainly suitable for longer programmes with simple graphic requirements.You can display all symbols on the keyboard in 121 different colours, either reverse or flashing. By choosing the right symbols, many things can be displayed easily and quickly e.g. playing cards, rectangles etc.

Normally these graphics are programmed in BASIC with PRINT commands, consisting of the graphic symbols and the colour etc. In some cases it may be more convenient (or unavoidable) to write the required characters directly into the graphics store of the C-16. If for example you try to write a character at bottom right hand corner of the screen (line 25,column 40), you will find that the screen scrolls back to the top after the PRINT command, and the bottom right-hand corner becomes free again. This problem, however, can be solved easily and quickly with a direct entry into the graphics store.

Let us first have a look at how the C-16 administers the screen internally. Two memory spaces belong to each of the 1000 (40x25) representable characters in the RAM of the C-16. In one is the internal sign code (the so-called Video-RAM) in the other one the colour (Colour-RAM). The Video-RAM is in the memory from 3072 onwards, the Colour-RAM from 2048 onwards; the code of the first character (top left, so-called HOME-position) is stored at 3072 and the colour at 2048. The next code in Ram is for the character to the right of the first position and so on to the end of the first line. The easiest way to illustrate this is with the following summary:

VIDEO-RAM (3072-4071)

Column:	1	2	3	...	38	39	40
Line:							
1	3072	3073	3074	...	3109	3110	3111
2	3112						3151
3	3152						3191
.	.						.
.	.						.
.	.						.
23	3952						3991
24	3992						4031
25	4032	4033	4034	...	4069	4070	4071

COLOUR-RAM (2048-3047)

Column	1	2	3	...	38	39	40
Line:							
1	2048	2049	2050	...	2085	2086	2087
2	2088						2127
3	2128						2167
.	.	.					.
.	.						.
.	.						.
23	2928						2967
24	2968						3007
25	3008	3009	3010	...	3045	3046	3047

You will find the coding of the individual characters in the appendix of your manual. The letter A is stored in Video-RAM as value 1. The value which is stored in the Colour RAM, consists of the colour, the brightness and the information whether the character is to be flashing or not. The following short programme erases the screen and writes a red flashing heart at the bottom right corner of the screen:

```
100 SCNCLR
110 POKE 4071,83
120 POKE 3047,184
```

The current cursor position is not changed by direct writing on to the screen and colour memory; also the unwelcome scrolling (see above) ceases.

2.2 THE GRAPHIC POSSIBILITIES OF TED

In this chapter, we will mainly deal with the TED registers which are responsible for the graphics. Before we go into the details, I will give you an idea of the graphic possibilities the TED-chip of the C-16 offers:

* Free programmable character set with 256 characters

* Creation of reverse characters by software or hardware

* Standard- and Multicolour Characters

* Extended Colour Mode

* Standard Hires Bitmap Mode

* Multicolour Bitmap Mode

* Programmable Scanning Interrupt

In the following chapters we will discuss all these possibilities and explain them with example programmes.

2.3 FREE PROGRAMMABLE CHARACTER SET

As you have already learnt in chapter 2.1 the easiest and least memory space consumicccccccphics possibility of the C-16 is the use of the existing graphics characters. With this you can display max. 128 characters as well as further 128 reverse characters, totalling 256. Unfortunately you may not often find the required character even among these 256 and might ask yourself if it is possible to define a character yourself. The answer in short is YES - it is possible! because you can tell the TED-chip where to get it's information for creating own characters.

Normally TED receives its information for the character set from a special ROM, the so-called Character-ROM, whiich contains the standard characters. If you now tell the TED-Chip to retrieve its information from RAM, you can define your own characters. Luckily the TED-Chip offers two possibilities for a character set of your own: firstly you can define the complete character set, i.e. 256 new characters or it is possible to only define 128 characters new and the reverse characters can be created by the TED-Chip itself. The latter has the big advantage that storing space is saved which is very limited in our C-16. Therefore instead of 2K RAM for a complete 256 character set,we only need 1K RAM for 128 characters.

Let us therefore mainly deal with the second possibility because with 128 self-defined characters almost every problem can be solved. Let's have a look how a character is made up: it always consists of a Matrix of 8x8 points. If we look at the letter A it gives the following picture:

```
...**...
..****..
.**..**.
.******.
.**..**.
.**..**.
.**..**.
........
```

An asterisk * always stands for a visible, and a point . for an

invisible pixel. Within this 8x8 Matrix we can create all sorts of characters. Before creating and trying out new characters it is advisable to first copy the existing characters from ROM into RAM and then replace some of the characters which won't be needed with your own characters. In the following chapter, first the character set is copied into RAM and then the relevant area protected from overwriting by a BASIC Programme. The "@" sign of the standard ch. set is replaced by the degree symbol and demonstrated in a short sentence. Please key in the following example:

```
100 POKE 55,0 : POKE 56,60 : CLR
110 POKE 1177,62
120 FOR I=0 TO 1023
130 : POKE 60*256+I,PEEK(53248+I)
140 NEXT I
150 POKE 1177,63
160 POKE 65299,60
170 POKE 65298,192
180 POKE 1351,128
190 FOR I=0 TO 7
200 : READ A
210 : POKE 60*256+I,A
220 NEXT I
230 SCNCLR
240 CHAR 1,6,12,"TEMPERATURE IS 24@C"
250 CHAR 1,2,23,"PLEASE KEY FOR NORMAL CHARACTER SET"
260 GETKEY K$
270 POKE 65299,208
280 POKE 65298,196
290 POKE 1351,0
300 DATA 24,36,36,24,0,0,0,0
```

Now a line by line explanation of this programme:

100	Reserve memory space for character set
110	Switch PEEK-routine to ROM-Read
120-140	Copy character set (1K) into RAM
150	Switch PEEK-routine to RAM-Read
160	Character-Set address to 60*256 = 15360 ($3C00 hex)
170	Read character set from RAM instead of ROM
180	Switch-off CBM/SHIFT key (make it non-optional)

190-220 Poke character definition into RAM
230 Erase screen
240-250 Print text on screen
260 Wait for key

*If you press any key, your degree symbol is replaced by @ from normal ch. set.

270-280 Back to normal character set
290 Switch-on CBM/SHIFT key
300 Character definition for degree symbol. This data is used by lines 190-220.

2.4 HIGH RESOLUTION GRAPHICS

When writing games, drawing schedules for business use or writing other programmes, you will sooner or later require a high resolution screen presentation.

The Commodore C-16 is tailor made for this: High Resolution is possible by "Bit-Mapping" the screen. "Bit-Mapping" is the method whereby each pixel on the screen gets its own Bit in the memory. If this memory bit is one, the corresponding pixel is switched on. If bit is a zero, the pixel is switched off.

Working with high resolution graphics has, however, a few disadvantages and is therefore not always used. By Bit-Mapping the whole screen, considerable memory space is taken up. Each pixel requires a memory bit , i.e. you need 1 Byte for 8 pixel. Since each character consists of a 8x8 Matrix and 40 lines x 25 columns are available , the resolution is 320 x 200 pixel. That makes 64000 pixel, of which each requires a memory bit. For Bit-Mapping the whole screen you will therefore need 8000 Bytes.

In order to understand the disadvantages of the Hires-Mode, please key in the following example. You will see that you get unwanted colour effects. This happens because the C-16 can in Hires-Mode only show one foreground and one background colour in each 8x8 field. If you try to write on an already used 8x8 field again it changes colour. This is exactly what our demo does:

```
100 GRAPHIC 1,1
110 FOR X=0 TO 190 STEP 10
120 : COLOR 1, ((X/10) AND15)+1,5
130 BOX 1,X,X,X+16,X+10,,1
140 NEXT X
150 FOR X=0 TO 192 STEP 8
160 : COLOR 1,((X/8)AND15)+1,5
170 : BOX 1,X+64,X,X+80,X+8,,1
180 NEXT X
190 COLOR 1,1,0 : CHAR 1,1,20"KEY PLEASE"
200 GETKEY K$: GRAPHIC 0
```

A possible remedy for this problem is supplied by the Multicolour-Mode which will be discussed in the next chapter. As an example for the Hires-Mode we will present you with a Mini-Drawing-Program in machine language. All those, who know little or nothing about machine language, should now study chapter 4 before starting with the following program. If you are familiar with computer language and the C16 built-in monitor as far as being able to key in the program and store it , then you should have no problem with the following program. We have provided a disassembled listing for easier understanding of the program. You can start the program in machine language by: G1000. Please remember to SAVE the machine program first before trying it out! The operation of the drawing program is very simple: you can draw lines in all directions with the joystick (in port 2). Keeping the FIRE button pressed on the joystick, you can erase the lines again by going over them. You can erase your whole work of art by using the SPACE bar.
For advanced machine language programmers, this program can be used as a starting point for own creations.
Dont' be shy, the C-16 can't do more than crash.

```
>1000 00 0B 10 00 00 9E 34 31 :.......4

>1008 31 32 00 00 00 00 00 00 :12......

>1010 A9 36 8D 06 FF A9 C8 8D :)6.)h.

>1018 12 FF 20 EA 10 A2 00 A9 : j.".)

>1020 33 9D 00 08 9D 00 09 9D :3......

>1028 00 0A 9D 00 0B A9 05 9D :....)..

>1030 00 0D 9D 00 0D 9D 00 0E :r........
>1038 9D 00 0F CA D0 E1 86 D6 :r...jp!.v
>1040 A9 A0 85 D5 D9 64 85 D4 :r) .u)$.t
>1048 E8 86 D2 20 E4 FF C9 20 :r(.r $?i
>1050 D0 03 20 EA 10 20 C5 10 :rp. *. e.
>1058 D0 02 E6 D2 A5 D0 F0 33 :rp.&r%p03
>1060 30 1C A5 D5 18 69 01 85 :r0.%u.)..
>1068 D5 AA A5 D6 69 00 85 D6 :ru*%v)..v
>1070 F0 21 E0 40 90 1D A9 00 :r0! @..).
```

```
>1078 85 D5 85 D6 F0 15 38 A5 :r.u.v0.8%
>1080 D5 E9 01 85 D5 A5 D6 E9 :ru)..u%v)
>1088 00 B0 06 A9 3f 85 D5 A9 :r.0.)?.u)
>1090 01 85 D6 A5 D1 F0 1B 30 :r..v%q0.0
>1098 0E A4 D4 C8 C0 C8 D0 02 :r.$th@hp.
>10A0 A0 00 84 D4 4C B2 10 A5 :r ..tl2.%
>10A8 D4 38 E9 01 B0 02 A9 C7 :rt8).0.)g
>10B0 85 D4 A5 D2 4A 20 02 11 :r.t%rj ..
>10B8 A2 0F A0 FF 88 D0 FD CA :r". ?.p=j
>10C0 D0 F8 4C 4B 10 78 A9 FD :rp8lk.8)=
>10C8 8D 08 FF AD 08 FF A0 00 :r..?-.? .
>10D0 A2 00 4A B0 01 88 4A B0 :r".j0..j0
>10D8 01 C8 4A B0 01 CA 4A B0 :r.hj0.jj0
>10E0 01 E8 86 D0 84 D1 29 08 :r.(.p.q).
>10E8 58 60 A9 00 85 D7 A9 20 :rx )..w)
>10F0 85 D8 A2 20 A0 00 98 91 :r.x"  ...
>10F8 D7 88 D0 FB E6 D8 CA D0 :rw.p;&xjp
>1100 F6 60 B0 0B 20 1A 11 BD :r6 0. ..=
>1108 64 11 31 D7 91 D7 60 20 :r$.1w.w
>1110 1A 11 BD 5C 11 11 D7 91 :r..=£..w.
>1118 D7 60 A5 D4 29 07 A8 A5 :rw %t).(%
>1120 D4 29 F8 85 D7 A9 00 06 :rt)8.w)..
>1128 D7 2A 06 D7 2A 06 D7 2A :rw*.w*.w*
>1130 85 D8 85 D3 A5 D7 0A 26 :r.x.s%w.&
>1138 D3 0A 26 D3 65 D7 85 D7 :rs.&s%w.w
>1140 A5 D8 65 D3 85 D8 A5 D5 :r%x%s.x%u
>1148 29 07 AA A5 D5 29 F8 65 :r).*%u)8%
>1150 D7 85 D7 A5 D6 65 D8 69 :rw.w%v%x)
>1158 20 85 D8 60 80 40 20 10 :r .x .@ .
>1160 08 04 02 01 7F BF DF EF :r....??_/
>1168 F7 FB FD FE DB 20 37 20 :r7;=>[ 7
```

Now SAVE the programme on a cassette or disc by the command.

```
S"HIGHRESDRAW",01,1001,116C   (Cass.)
S"HIGHRESDRAW",08,1001,116C   (Disc)
```

```
. 1010  A9 36    LDA #$36
. 1012  8D 06 FF STA $FF06
. 1015  A9 C8    LDA #$C8
. 1017  8D 12 FF STA $FF12
. 101A  20 EA 10 JSR $10EA
. 101D  A2 00    LDX #$00
```

```
. 101F  A9 33     LDA #$33
. 1021  9D 00 08  STA $0800,x
. 1024  9D 00 09  STA $0900,x
. 1027  9D 00 0A  STA $0A00,x
. 102A  9D 00 0B  STA $0B00,x
. 102D  A9 05     LDA #$05
. 102F  9D 00 0C  STA $0C00,x
. 1032  9D 00 0D  STA $0D00,x
. 1035  9D 00 0E  STA $0E00,x
. 1038  9D 00 0F  STA $0F00,x
. 103B  CA        DEX
. 103C  D0 E1     BNE $101F
. 103E  86 D6     STX $D6
. 1040  A9 A0     LDA #$A0
. 1042  85 D5     STA $D5
. 1044  A9 64     LDA #$64
. 1046  85 D4     STA $D4
. 1048  E8        INX
. 1049  86 D2     STX $D2
. 104B  20 E4 FF  JSR $FFE4
. 104E  C9 20     CMP #$20
. 1050  D0 03     BNE $1055
. 1052  20 EA 10  JSR $10EA
. 1055  20 C5 10  JSR $10C5
. 1058  D0 02     BNE $105C
. 105A  E6 D2     INC $D2
. 105C  A5 D0     LDA $D0
. 105E  F0 33     BEQ $1093
. 1060  30 1C     BMI $107E
. 1062  A5 D5     LDA $D5
. 1064  18        CLC
. 1065  69 01     ADC #$01
. 1067  85 D5     STA $D5
. 1069  AA        TAX
. 106A  A5 D6     LDA $D6
. 106C  69 00     ADC #$00
. 106E  85 D6     STA $D6
. 1070  F0 21     BEQ $1093
. 1072  E0 40     CPX #$40
. 1074  90 1D     BCC $1093
. 1076  A9 00     LDA #$00
. 1078  85 D5     STA $D5
```

```
. 107A  85 D6     STA $D6
. 107C  F0 15     BEQ $1093
. 107E  38        SEC
. 107F  A5 D5     LDA $D5
. 1081  E9 01     SBC #$01
. 1083  85 D5     STA $D5
. 1085  A5 D6     LDA $D6
. 1087  E9 00     SBC #$00
. 1089  B0 06     BCS $1091
. 108B  A9 3F     LDA #$3F
. 108D  85 D5     STA $D5
. 108F  A9 01     LDA #$01
. 1091  85 D6     STA $D6
. 1093  A5 D1     LDA $D1
. 1095  F0 1B     BEQ $10B2
. 1097  30 0E     BMI $10A7
. 1099  A4 D4     LDY $D4
. 109B  C8        INY
. 109C  C0 C8     CPY #$C8
. 109E  D0 02     BNE $10A2
. 10A0  A0 00     LDY #$00
. 10A2  84 D4     STY $D4
. 10A4  4C B2 10  JMP $10B2
. 10A7  A5 D4     LDA $D4
. 10A9  38        SEC
. 10AA  E9 01     SBC #$01
. 10AC  B0 02     BCS $10B0
. 10AE  A9 C7     LDA #$C7
. 10B0  85 D4     STA $D4
. 10B2  A5 D2     LDA $D2
. 10B4  4A        LSR
. 10B5  20 02 11  JSR $1102
. 10B8  A2 0F     LDX #$0F
. 10BA  A0 FF     LDY #$FF
. 10BC  88        DEY
. 10BD  D0 FD     BNE $10BC
. 10BF  CA        DEX
. 10C0  D0 F8     BNE $10BA
. 10C2  4C 4B 10  JMP $104B
. 10C5  78        SEI
. 10C6  A9 FD     LDA #$FD
. 10C8  8D 08 FF  STA $FF08
```

```
. 10CB  AD 08 FF LDA $FF08
. 10CE  A0 00    LDY #$00
. 10D0  A2 00    LDX #$00
. 10D2  4A       LSR
. 10D3  B0 01    BCS $10D6
. 10D5  88       DEY
. 10D6  4A       LSR
. 10D7  B0 01    BCS $10DA
. 10D9  C8       INY
. 10DA  4A       LSR
. 10DB  B0 01    BCS $10DE
. 10DD  CA       DEX
. 10DE  4A       LSR
. 10DF  B0 01    BCS $10E2
. 10E1  E8       INX
. 10E2  86 D0    STX $D0
. 10E4  84 D1    STY $D1
. 10E6  29 08    AND #$08
. 10E8  58       CLI
. 10E9  60       RTS
. 10EA  A9 00    LDA #$00
. 10EC  85 D7    STA $D7
. 10EE  A9 20    LDA #$20
. 10F0  85 D8    STA $D8
. 10F2  A2 20    LDX #$20
. 10F4  A0 00    LDY #$00
. 10F6  98       TYA
. 10F7  91 D7    STA ($D7),Y
. 10F9  88       DEY
. 10FA  D0 FB    BNE $10F7
. 10FC  E6 D8    INC $D8
. 10FE  CA       DEX
. 10FF  D0 D6    BNE $10F7
. 1101  60       RTS
. 1102  B0 0B    BCS $110F
. 1104  20 1A 11 JSR $111A
. 1107  BD 64 11 LDA $1164,X
. 110A  31 D7    AND ($D7),Y
. 110C  91 D7    STA ($D7),Y
. 110E  60       RTS
. 110F  20 1A 11 JSR $111A
. 1112  BD 5C 11 LDA $115C,X
```

```
. 1115  11 D7    ORA ($D7),Y
. 1117  91 D7    STA ($D7),Y
. 1119  60       RTS
. 111A  A5 D4    LDA $D4
. 111C  29 07    AND #$07
. 111E  A8       TAY
. 111F  A5 D4    LDA $D4
. 1121  29 F8    AND #$F8
. 1123  85 D7    STA $D7
. 1125  A9 00    LDA #$00
. 1127  06 D7    ASL $D7
. 1129  2A       ROL
. 112A  06 D7    ASL $D7
. 112C  2A       ROL
. 112D  06 D7    ASL $D7
. 112F  2A       ROL
. 1130  85 D8    STA $D8
. 1132  85 D3    STA $D3
. 1134  A5 D7    LDA $D7
. 1136  0A       ASL
. 1137  26 D3    ROL $D3
. 1139  0A       ASL
. 113A  26 D3    ROL $D3
. 113C  65 D7    ADC $D7
. 113E  85 D7    STA $D7
. 1140  A5 D8    LDA $D8
. 1142  65 D3    ADC $D3
. 1144  85 D8    STA $D8
. 1146  A5 D5    LDA $D5
. 1148  29 07    AND #$07
. 114A  AA       TAX
. 114B  A5 D5    LDA $D5
. 114D  29 F8    AND #$F8
. 114F  65 D7    ADC $D7
. 1151  85 D7    STA $D7
. 1153  A5 D6    LDA $D6
. 1155  65 D8    ADC $D8
. 1157  69 20    ADC #$20
. 1159  85 D8    STA $D8
. 115B  60       RTS

>115C 80 40 20 10 08 04 02 01 :r.@ ....
```

2.5 MULTICOLOUR-GRAPHICS

Because of the standard high resolution graphics, you can even address individual pixels on the screen. There are two values for each pixel in the character memory: a 1 for "on" and 0 for "off". If a pixel has the value 1, it will be shown in the colour chosen by you at the chosen screen position. With high resolution graphics all pixels within an 8x8 matrix can be shown either in the foreground or background colour. Thus the colour resolution within this area is limited. There could for example be problems if two lines with different colours cross (see last example in previous chapter).
This problem is solved by the Multicolour-Mode. Each pixel can have one of four colours: screen colour (background register 0,65301), background colour 1 (65302), background colour 2 (65303) or the character colour. The only restriction is in the horizontal resolution, since in the Multicolour-Mode each pixel is twice as wide as in Hires-Graphics. The advantages of the Multicolour-Mode, however,prevail by far. Here is a first example:

```
100 GRAPHIC 3,1
110 COLOR 0,1,0 : COLOR 1,3,0 : COLOR 2,8,7 : COLOR 3,13,4
120 FOR C=3 to 1 STEP -1
130 : FOR A=0 TO 180 STEP 10
140 :   CIRCLE C,20+C*30,100,10,50,,,A,20
150 : NEXT A
160 NEXT C
170 COLOR 1,2,7 : CHAR 1,1,20,"PLEASE KEY"
180 GETKEY K$ : DRAW 0,0,100 to 159,100
190 GETKEY K$ : GRAPHIC 0
```

EXPLANATION LINE BY LINE:

110 Definition of background colour 0, foreground colour,background colours 1 & 2

120 Draw in each of the three colours

130 Draw ovals at different angles

140 Draw oval

150 Next angle

160 Next colour

170 Character colour on white, print text
180 Wait for key. Draw line.

190 Wait for key. Back to normal text mode.

As you can see the three circles are drawn in different colours without showing after-colourings. This effect appears only if you key in another colour. In practice (and in most commercial games) mostly this graphic mode is mostly used. It among other things also allows for the programming of Software-Sprites (more about this later).

Because of the limited memory it is also useful in this mode to work with self-defined characters. But you have to consider that text fade-ins are not so easily possible since the integral character set in the C-16 is designed for the normal mode. That is why the letters often look a bit odd. This can be avoided by a character set of your own. In contrast to the normal mode the multicolour characters do not use a Bit for a pixel each, but 2 Bits determine the colour. Therefore only 4 pixel are available widthwise (hence the resolution of 160x200). The coordination of the colours to the Bit combinations table is shown below.

BITS	COLOUR SOURCE	
00	Background 0	(65301)
01	Background 1	(65302)
10	Background 2	(65303)
11	Foreground	

Let's look at the following self defined character, where we have chosen as background colour 0 (H0) black, background colour 1 (H1) red, background colour 2 (H2) yellow and foreground colour (V0) green:

BIT SAMPLE	COLOUR SOURCE	COLOURS
00 01 10 00	H0 H1 H2 H0	black red yellow black
00 11 11 00	H0 V0 V0 H0	black green green black
01 10 01 10	H1 H2 H1 H2	red yellow red yellow
01 11 11 10	H1 V0 V0 H2	red green green yellow
01 10 01 10	H1 H2 H1 H2	red yellow red yellow
01 10 01 10	H1 H2 H1 H2	red yellow red yellow
01 10 01 10	H1 H2 H1 H2	red yellow red yellow
00 00 00 00	H0 H0 H0 H0	black black black black

The following example copies again the normal character set into RAM and then creates own Multicolour-Characters for a butterfly:

```
100 POKE 55,0 : POKE 56,60 : CLR
110 POKE 1177,62
120 FOR I=0 TO 1023
130 : POKE 60*256+I,PEEK(53248+I)
140 NEXT I
150 POKE 1177,63
160 POKE 65299,60
170 POKE 65298,192
180 POKE 65287,24
190 POKE 1351,128
```

```
200 POKE 65302,0 : POKE 65303,93
210 FOR I=0 TO 15
220 : READ A
230 : POKE 60*256+8*40+I,A
240 NEXT I
250 SCNCLR
260 PRINT TAB(5)CHR$(154)"HALLO BUTTERFLY"
270 PRINT:PRINT TAB(10)"()"SPC(10)CHR$(153)"()"
280 CHAR 1,2,23,"KEY FOR NORMAL CHARACTER SET"
290 GETKEY K$
300 POKE 65299,208
310 POKE 65298,196
320 POKE 65287,8
330 POKE 1351,0
340 DATA 196,241,237,237,253,253,241,193
350 DATA 76,60,236,236,252,252,60,12
```

EXPLANATION LINE BY LINE

100 Reserve memory space for character set

110 Switch PEEK-routine to ROM-read

120-140 Copy character set (1K) into RAM

150 Switch PEEK-routine to RAM-read

160 Character set address to 60*256 = 15360 ($3C00)

170 Read character set out of RAM instead of ROM

180 Switch-on Multicolour-Mode

190 Switch-off CBM/SHIFT key

200 Background colour 1 on black, 2 on light blue no. 5

210-240 Poke character definition of butterfly into RAM

250 Erase screen

260-280 Print text and butterflies

290 Wait for key

300-320 Restore standard values

330 Switch-on CBM/SHIFT key

340-350 Character definition for Multicolour butterfly

Finally two examples for animated graphics as well as a Multicolour version of the mini drawing program. First we alter the previous butterfly demo by defining 3 different butterflies and portray these alternately, i.e. in the sequence: 1,2,3,2,1,2,3,2,1,... thereby creating the impression that the butterfly is moving.

The 2nd example shows the use of the SHAPE-commands of the C-16 which allow a Sprite-like animation in the Multicolour-Mode.

The 3rd example is a Multicolour-variant of the mini drawing program which you saw in the previous chapter. This time you can choose between the 3 Multicolour-Colours by pressing a button and clear the screen by using the space bar.

```
100 POKE 55,0 : POKE 56,60 : CLR
110 POKE 1177,62
120 FOR I=0 TO 1023
130 : POKE 60*256+I,PEEK(53248+I)
140 NEXT I
150 POKE 1177,63
160 POKE 65299,60
170 POKE 65298,192
180 POKE 65287,24
190 POKE 1351,128
200 POKE 65302,0 : POKE 65303,93
210 FOR I=0 TO 47
220 : READ A
230 : POKE 60*256+8*40+I,A
240 NEXT I
250 SCNCLR
260 PRINT TAB(6)CHR$(154)"HALLO BUTTERFLY"
270 CHAR 1,1,23,"KEY FOR NORMAL CHARACTER SET"
280 Y=24 : Y1=24
290 GET K$ : IF K$<>"" THEN PRINT CHR$(144) : goto 360
300 A$="()" : GOSUB 410
310 A$="*+" : GOSUB 410
320 A$=",-" : GOSUB 410
330 A$="*+" : GOSUB 410
340 GOTO 290
350 GETKEY K$
360 POKE 65299,208
370 POKE 65298,196
380 POKE 65287,8
390 POKE 1351,0
400 END
410 CHAR 1,12,Y1,"  "
420 CHAR 1,12,Y,CHR$(154)+A$
430 CHAR 1,18,Y1,"  "
440 CHAR 1,18,Y,CHR$(153)+A$
450 Y1=Y : Y=Y-1 : IF Y<0 THEN Y=24
460 RETURN
470 DATA 196,241,237,237,253,253,241,193
480 DATA 76,60,236,236,252,252,60,12,52,49,57,61,61,49,49,49
500 DATA 112,48,176,240,240,48,48,48,4,3,3,3,3,3,3,3
520 DATA 64,0,0,0,0,0,0,0
```

Explanation line by line:

100	Reserve memory space for character set
110	Switch PEEK-routine to ROM-read
120-140	Copy character set (1K) into RAM
150	Switch PEEK-routine to RAM-read
160	Character set address at 60*256 = 15360 ($3C00)
170	Read character set from RAM instead of ROM
180	Switch-on Multicolour-Mode
190	Switch-off CBM/SHIFT key
200	Background colour 1 : black, 2 : light blue no. 5
210-240	Poke character definition of butterfly into RAM
250	Erase screen
260-270	Print text
280	Start of butterflies on bottom screen edge
290	Wait for key. If yes, finish program.
300	1. Create variant of butterfly
310	2. " " " "
320	3. " " " "
330	4. " " " "
340	Back to start of loop
350-400	Normal colours, end of program

410 Write two spaces in previous position of right butterfly

420 Colour blue, show right butterfly

430 Write two spaces in previous position of left butterfly

440 Colour green, show left butterfly

450 Move Y1 one line.If at top,then start again at bottom.

460 Sub-program finish

470-520 Definition of the 3 variants of butterfly.

Here is the 3rd example, the mini drawing program, which you already know from the Hires chapter, in Multicolour:

```
>1000 00 0B 10 00 00 9E 34 31 :r......41
>1008 31 32 00 00 00 00 00 00 :r12......
>1010 A9 36 8D 06 FF A9 18 8D :r)6..?)..
>1018 07 FF A9 C8 8D 12 FF 20 :r.?)h..?
>1020 CE 10 A2 00 A9 33 9D 00 :rn.".)3..
>1028 08 9D 00 09 9D 00 0A 9D :r........
>1030 00 0B A9 52 9D 00 0C 9D :r..)r....
>1038 00 0D 9D 00 0E 9D 00 0F :r........
>1040 CA D0 E1 A9 30 8D 15 FF :rjp!)0..?
>1048 A9 77 8D 16 FF E8 86 D2 :r)7..?(.r
>1050 A9 50 85 D6 A9 64 85 D5 :r)p.v)$.u
>1058 20 E4 FF C9 20 D0 03 20 :r $?i p.
>1060 CE 10 20 A9 10 D0 08 E6 :rn. ).p.&
>1068 D2 A5 D2 29 03 85 D2 A5 :rr%r)..r%
>1070 D0 18 65 D6 c9 FF D0 04 :rp.%vi?p.
>1078 A9 9F D0 06 C9 A0 90 02 :r).p.i ..
>1080 A9 00 85 D6 A5 D1 18 65 :r)..v%q.%
>1088 D5 C9 FF D0 04 A9 C7 D0 :rui?p.)gp
>1090 06 C9 C8 90 02 A9 00 85 :r.ih..)..
>1098 D5 20 E6 10 A2 0F A0 FF :ru &.". ?
```

```
>10A0 88 D0 FD CA D0 F8 4C 58 :r.p=jp8lx
>10A8 10 78 A9 FD 8D 08 FF AD :r.8)=..?-
>10B0 08 FF A0 00 A2 00 4A B0 :r.? ."."j0
>10B8 01 88 4A B0 01 C8 4A B0 :r..j0.hj0
>10C0 01 CA 4A B0 01 E8 86 D0 :r.jj0.(.p
>10C8 84 D1 29 08 58 60 A9 00 :r.q).x ).
>10D0 85 D7 A9 20 85 D8 A2 20 :r.w) .x"
>10D8 A0 00 98 91 D7 88 D0 FB :r ...w.p;
>10E0 E6 D8 CA D0 F6 60 A5 D5 :r&xjp6 %u
>10E8 29 07 A8 A5 D5 29 f8 85 :r).(%u)8.
>10F0 D7 A9 00 85 D4 06 D7 2A :rw)..t.w*
>10F8 06 D7 2A 06 D7 2A 85 D8 :r.w*.w*.x
>1100 85 D3 A5 D7 0A 26 D3 0A :r.s%w.&s.
>1108 26 D3 65 D7 85 D7 A5 D8 :r&s%w.w%x
>1110 65 D3 85 D8 A5 D6 29 FC :r%s.x%v)<
>1118 0A 26 D4 65 D7 85 D7 A9 :r.&t%w.w)
>1120 20 65 D8 65 D4 85 D8 A5 :r %x%t.x%
>1128 D6 29 03 AA B1 D7 3D 42 :rv).*1w=b
>1130 11 85 D3 BD 42 11 49 FF :r..s=b.i?
>1138 A6 D2 3D 46 11 05 D3 91 :r&r=f..s.
>1140 D7 60 3F CF F3 FC 00 55 :rw ?o3<.u
>1148 AA FF AA AA AA AA AA AA : * * * *9
```

SAVE the prog. on Cass. or Disc

```
S"MULTIDRAW",01,1001,114A    (Cass.)

S"MULTIDRAW",08,1001,114A    (Disc)
```

```
. 1010  A9 36    LDA #$36
. 1012  8D 06 FF STA $FF06
. 1015  A9 18    LDA #$18
. 1017  8D 07 FF STA $FF07
. 101A  A9 C8    LDA #$C8
. 101C  8D 12 FF STA $FF12
. 101F  20 CE 10 JSR $10CE
. 1022  A2 00    LDX #$00
. 1024  A9 33    LDA #$33
. 1026  9D 00 08 STA $0800,X
```

```
. 1029  9D 00 09 STA $0900,X
. 102C  9D 00 0A STA $0A00,X
. 102F  9D 00 0B STA $0B00,X
. 1032  A9 52    LDA #$52
. 1034  9D 00 0C STA $0C00,X
. 1037  9D 00 0D STA $0D00,X
. 103A  9D 00 0E STA $0E00,X
. 103D  9D 00 0F STA $0F00,X
. 1040  CA       DEX
. 1041  D0 E1    BNE $1024
. 1043  A9 30    LDA #$30
. 1045  8D 15 FF STA $FF15
. 1048  A9 77    LDA #$77
. 104A  8D 16 FF STA $FF16
. 104D  E8       INX
. 104E  86 D2    STX $D2
. 1050  A9 50    LDA #$50
. 1052  85 D6    STA $D6
. 1054  A9 64    LDA #$64
. 1056  85 D5    STA $D5
. 1058  20 E4 FF JSR $FFE4
. 105B  C9 20    CMP #$20
. 105D  D0 03    BNE $1062
. 105F  20 CE 10 JSR $10CE
. 1062  20 A9 10 JSR $10A9
. 1065  D0 08    BNE $106F
. 1067  E6 D2    INC $D2
. 1069  A5 D2    LDA $D2
. 106B  29 03    AND #$03
. 106D  85 D2    STA $D2
. 106F  A5 D0    LDA $D0
. 1071  18       CLC
. 1072  65 D6    ADC $D6
. 1074  C9 FF    CMP #$FF
. 1076  D0 04    BNE $107C
. 1078  A9 9F    LDA #$9F
. 107A  D0 06    BNE $1082
. 107C  C9 A0    CMP #$A0
. 107E  90 02    BCC $1082
. 1080  A9 00    LDA #$00
. 1082  85 D6    STA $D6
. 1084  A5 D1    LDA $D1
```

```
. 10D0  85 D7    STA $D7
. 10D2  A9 20    LDA #$20
. 10D4  85 D8    STA $D8
. 10D6  A2 20    LDX #$20
. 10D8  A0 00    LDY #$00
. 10DA  98       TYA
. 10DB  91 D7    STA ($D7),Y
. 10DD  88       DEY
. 10DE  D0 FB    BNE $10DB
. 10E0  E6 D8    INC $D8
. 10E2  CA       DEX
. 10E3  D0 F6    BNE $10DB
. 10E5  60       RTS
. 10E6  A5 D5    LDA $D5
. 10E8  29 07    AND #$07
. 10EA  A8       TAY
. 10EB  A5 D5    LDA $D5
. 10ED  29 F8    AND #$F8
. 10EF  85 D7    STA $D7
. 10F1  A9 00    LDA #$00
. 10F3  85 D4    STA $D4
. 10F5  06 D7    ASL $D7
. 10F7  2A       ROL
. 10F8  06 D7    ASL $D7
. 10FA  2A       ROL
. 10FB  06 D7    ASL $D7
. 10FD  2A       ROL
. 10FE  85 D8    STA $D8
. 1100  85 D3    STA $D3
. 1102  A5 D7    LDA $D7
. 1104  0A       ASL
. 1105  26 D3    ROL $D3
. 1107  0A       ASL
. 1108  26 D3    ROL $D3
. 110A  65 D7    ADC $D7
. 110C  85 D7    STA $D7
. 110E  A5 D8    LDA $D8
. 1110  65 D3    ADC $D3
. 1112  85 D8    STA $D8
. 1114  A5 D6    LDA $D6
. 1116  29 FC    AND #$FC
. 1118  0A       ASL
```

```
. 1119  26 D4     ROL $D4
. 111B  65 D7     ADC $D7
. 111D  85 D7     STA $D7
. 111F  A9 20     LDA #$20
. 1121  65 D8     ADC $D8
. 1123  65 D4     ADC $D4
. 1125  85 D8     STA $D8
. 1127  A5 D6     LDA $D6
. 1129  29 03     AND #$03
. 112B  AA        TAX
. 112C  B1 D7     LDA ($D7),Y
. 112E  3D 42 11  AND $1142,X
. 1131  85 D3     STA $D3
. 1133  BD 42 11  LDA $1142,X
. 1136  49 FF     EOR #$FF
. 1138  A6 D2     LDX $D2
. 113A  3D 46 11  AND $1146,X
. 113D  05 D3     ORA $D3
. 113F  91 D7     STA ($D7),Y
. 1141  60        RTS

>1142 3F CF F3 FC 00 55 AA FF :r?o3<.u*?
```

2.6 EXTENDED COLOUR-GRAPHICS

In this unfortunately little known mode, you can use one of the four background colours for each character on the screen. It is for example possible to show a blue character with a yellow background on a white screen. This choice of colours has of course also its price: we can only use 64 different characters since the two top Bits of the character have to be used for definition of the background colour. The following chart should make this clear:

SCREEN CODE	CHARACTER	BACKGROUND-COLOUR REGISTER
0 - 63	0 - 63	65301
64 - 127	0 - 63	65302
128 - 191	0 - 63	65303
192 - 255	0 - 63	65304

Program below will show you the practical application of this mode:

```
100 SCNCLR
110 COLOR 0,1,2
120 COLOR 1,2,3
130 COLOR 2,4,5
140 COLOR 3,6,7
150 POKE 65286,PEEK(65286)OR64
160 FOR I=0 TO 63
170 : POKE 3072+I,I
180 : POKE 3136+I,I+64
190 : POKE 3200+I,I+128
200 : POKE 3264+I,I+192
210 NEXT I
220 I=0
```

```
230 FOR C=0 TO 15
240 : FOR L=0 TO 7
250 :   POKE 2048+I,L*16+C
260 :   POKE 2176+I,L*16+C
270 :   I=I+1
280 : NEXT L
290 NEXT C
300   CHAR   1,14,12,"PLEASE   KEY"    :    GETKEY    K$    :    POKE
65286,PEEK(65286)AND191
310 COLOR 0,2,7
```

EXPLANATION LINE BY LINE:

100 Erase screen

110-140 Choose background colours

150 Switch-on Extended Colour Mode

160 For each possible character:

170 Create one set with background colour 1

180 " " " " " " 2

190 " " " " " " 3

200 " " " " " " 4

210 Next character

220 Put variant 1 to 0

230-240 For each colour and brightness:

250-260 Write colour and brightness into colour memory

270 Increment pointer

280-290 Next brightness and colour

300 Print text, wait for key, normal mode.

310 Switch-on normal background colour (white).

2.7 SOFT SCROLLING

When the last line of the C-16's screen is filled with text, the whole picture is automatically scrolled one line up. That means the top line is removed and all other lines move towards the top by one position. This scrolling happens normally line by line but there are other situations where it would be desirable for this scrolling to be soft, i.e. only by one pixel.

The TED-Graphic Chip offers even this possibility and at least the vertical scrolling is programmable in BASIC. Firstly the no. of screen lines has to be reduced to 24 (Bit 3 of register 65286). The bottom 3 Bits of the same register define the scroll-position from 0 to 7. As soon as this value has reached 7, a completely new line becomes visible. We then have to set this value to 0 again and get the next line. The following BASIC-Program demonstrates this effect:

```
100 SCNCLR
110 FOR I=0 TO 24
120 : GOSUB 240
130 NEXT I
140 SR=65286
150 FOR T=1 TO 20 : NEXT T
160 POKE SR,(PEEK(SR)AND240)OR 7
170 GOSUB 240
180 FOR I=6 TO 0 STEP -1
190 : FOR T=1 TO 60 : NEXT T
200 : POKE SR,(PEEK(SR)AND240)OR I
210 NEXT I
220 GET K$ : IF K$="" THEN 150
230 POKE SR,(PEEK(SR)AND240)OR 11 : END
240 PRINT CHR$(13)"PRESS ANY KEY"; : RETURN
```

LINE BY LINE EXPLANATION:

100 Erase screen

110-130 Print text on screen

140 Put SR on address of Scroll-Register (vertical)

150 Time loop

160 Put on 24 lines and Scroll-Value on 7

170 Print message on presently invisible line 25

180 Decrement Scroll-Value

190 Time loop

200 Put Scroll-Value

210 Next value

220 Check key

230 Put on 25 lines again and normal Scroll-Value (3)

240 Subroutine, to print message

In the next chapter you will find another example of horizontal scrolling (in machine code), which together with the Raster Interrupt is responsible for good effects.

2.8 THE RASTER-INTERRUPT

The C-16 has the possibility to interrupt the processor dependant on the screen beam (raster) which creates the TV-picture. This can be used to write data on the screen while the raster beam is not on the screen, thus avoiding a flicker. Furthermore it is possible to subdivide the screen into different working areas. This is also used by the C-16 s BASIC in order to utilise the divided screen in graphic modes 2 and 4.

In order to create a Raster Interrupt you have to put Bit 1 of the Interrupt Enable Register ($FF0A) to 1. The lower 8 Bits of the line in which an interrupt is to be created has to be written into register $FF0B and the 9th Bit in Bit 0 of register $FF0A. As soon as the raster beam arrives at the selected line a raster interrupt is genrated and Bit 1 of the Raster Interrupt Status Register ($FF09) set to 1. To facilitate things you best look at the next example which creates a black band on the screen background.

```
>1000 00 0C 10 00 00 9E 34 31 :r......41
>1008 31 32 00 00 00 00 00 00 :r12......
>1010 78 A9 29 8D 14 03 A9 10 :r8))...).
>1018 8D 15 03 A9 02 8D 0A FF :r...)...?
>1020 A9 30 8D 0B FF 58 4C 26 :r)0..?xl&
>1028 10 AD 09 FF 8D 09 FF AD :r.-.?..?-
>1030 0B FF C9 50 90 0B A9 30 :r.?ip..)0
>1038 8D 0B FF EE 15 FF 4C 49 :r..?..?li
>1040 10 A9 50 8D 0B FF CE 15 :r.)p..?n.
>1048 FF 68 A8 68 AA 68 40 AA :r?(((*(@*
```

SAVE prog. on Cass. or Disc

```
S"RASTER-DEMO",01,1001,104F     (Cass.)

S"RASTER-DEMO",08,1001,104F     (Disc)
```

```
. 1010  78       SEI
. 1011  A9 29    LDA #$29
. 1013  8D 14 03 STA $0314
. 1016  A9 10    LDA #$10
. 1018  8D 15 03 STA $0315
. 101B  A9 02    LDA #$02
. 101D  8D 0A FF STA $FF0A
. 1020  A9 30    LDA #$30
. 1022  8D 0B FF STA $FF0B
. 1025  58       CLI
. 1026  4C 26 10 JMP $1026
. 1029  AD 09 FF LDA $FF09
. 102C  8D 09 FF STA $FF09
. 102F  AD 0B FF LDA $FF0B
. 1032  C9 50    CMP #$50
. 1034  90 0B    BCC $1041
. 1036  A9 30    LDA #$30
. 1038  8D 0B FF STA $FF0B
. 103B  EE 15 FF INC $FF15
. 103E  4C 49 10 JMP $1049
. 1041  A9 50    LDA #$50
. 1043  8D 0B FF STA $FF0B
. 1046  CE 15 FF DEC $FF15
. 1049  68       PLA
. 104A  a8       TAY
. 104B  68       PLA
. 104C  AA       TAX
. 104D  68       PLA
. 104E  40       RTI
```

Finally we would like to show you a program which shows the graphic possibilities of TED with Soft Scrolling and Raster Interrupt quite impressively. The combination of these two functions is used in many commercial games, e.g. to see a moving landscape. In our example we will put 3 texts onto the screen of which the middle one can be moved gently across the screen with the joystick - in both directions with variable speed. When you have analysed and understood this example, it shouldn't be too difficult anymore for you to write excellent graphic effects and programs of your own.

```
>1000 00 0B 10 00 00 9E 34 31 :.......41
>1008 31 32 00 00 00 00 00 00 :r12......
>1010 A9 CF 8D 15 FF A2 00 A9 :r)o..?".)
>1018 20 9D 00 0C 9D 00 0D 9D :r .......
>1020 00 0E 9D 00 0F A9 00 9D :r.....)..
>1028 00 08 9D 00 09 9D 00 0A :r........
>1030 9D 00 0B CA D0 E1 A0 12 :r...JP! .
>1038 B9 9F 11 99 82 0C 99 2A :r9......*
>1040 0F B9 B2 11 99 D2 0C 99 :r.92..r..
>1048 7A 0F B9 C5 11 99 C2 0D :r:.9E..B.
>1050 B9 D8 11 99 62 0E A9 26 :r9x..".)&
>1058 99 C2 09 99 62 0A 88 10 :r.B.."...
>1060 D7 86 D0 86 DB 20 8F 10 :rw.P.[ ..
>1068 20 B1 10 A9 FD 8D 08 FF :r 1.)=..?
>1070 AD 08 FF A2 00 4A 4A 4A :r-.?".JJJ
>1078 B0 01 CA 4A B0 01 E8 8A :r0.JJ0.(.
>1080 18 65 D0 C9 F7 F0 E1 C9 :r.%PI70!I
>1088 09 F0 DD 85 D0 D0 D9 78 :r.0].PPY8
>1090 A9 73 8D 14 03 A9 11 8D :r)3...)..
>1098 15 03 A9 02 8D 0A FF A9 :r..)...?)
>10A0 00 85 DA 85 D9 A9 5A 8D :r..z.Y)z.
>10A8 0B FF A9 00 8D 07 FF 58 :r.?)...?x
>10B0 60 A5 DB F0 FC A9 00 85 :r %[0<)..
>10B8 DB A5 D9 18 65 D0 A8 29 :r[%Y.%P()
>10C0 07 85 D9 98 29 08 D0 01 :r..Y.).P.
>10C8 60 A2 05 A9 0D 85 D2 85 :r ".)..r.
>10D0 D4 A9 09 85 D6 85 D8 A9 :rt)..v.x)
>10D8 90 85 D1 98 10 31 A0 FF :r..q..1 ?
>10E0 A5 D1 18 69 28 85 D1 85 :r%q.)(.q.
>10E8 D3 85 D5 85 D7 90 08 E6 :rs.u.w..&
>10F0 D2 E6 D4 E6 D6 E6 D8 E6 :rr&t&v&x&
>10F8 D3 E6 D7 C8 B1 D3 91 D1 :rs&whls.q
>1100 B1 D7 91 D5 C0 27 D0 F3 :rlw.u@'P3
>1108 CA D0 D3 84 DA F0 2D A0 :rJPs.z0-
>1110 26 A5 D1 18 69 28 85 D1 :r&%q.)(.q
>1118 85 D3 85 D5 85 D7 90 08 :r.s.u.w..
>1120 E6 D2 E6 D4 E6 D6 E6 D8 :r&r&t&v&x
>1128 E6 D3 E6 D7 B1 D1 91 D3 :r&s&wlq.s
>1130 B1 D5 91 D7 88 10 F5 CA :rlu.w..5J
>1138 D0 D5 86 DA A2 05 A4 DA :rPu.z".$z
```

```
>1140 A9 0D 85 D2 A9 09 85 D6 :r)..r)..v
>1148 A9 90 85 D1 85 D5 A5 D1 :r)..q.u%q
>1150 18 69 28 85 D1 85 D5 90 :r.)(.q.u.
>1158 04 E6 D2 E6 D6 98 49 27 :r.&r&v.I'
>1160 A8 B1 D1 85 DC B1 D5 A4 :r(1q.£1u$
>1168 DA 91 D5 A5 DC 91 D1 CA :rz.u%£.qJ
>1170 D0 DC 60 AD 09 FF 8D 09 :rP£ -.?..
>1178 FF AE 0B FF A9 40 E0 83 :r?..?)@ .
>1180 F0 08 05 D9 A2 83 86 DB :r0..Y"..[
>1188 D0 02 A2 5A 8E 0B FF 29 :rP."z..?)
>1190 07 EC 1D FF F0 FB 8D 07 :r.,.?0;..
>1198 FF 68 A8 68 AA 68 40 20 :r?(((*(@
>11A0 04 09 05 13 05 12 20 14 :R..... .
>11A8 05 09 0C 20 08 09 05 12 :R... ....
>11B0 20 20 20 20 20 02 0C 05 :R     ...
>11B8 09 02 14 20 13 14 05 08 :R... ....
>11C0 05 0E 20 20 20 16 05 12 :R..   ...
>11C8 13 03 08 09 05 02 05 20 :R.......
>11D0 04 01 13 20 08 09 05 12 :R... ....
>11D8 0D 09 14 20 05 09 0E 05 :R... ....
>11E0 0D 20 0A 0F 19 13 14 09 :R. ......
>11E8 03 0B 21 AA AA AA AA AA :R..!*****

. 1010  A9 CF    LDA #$CF
. 1012  8D 15 FF STA $FF15
. 1015  A2 00    LDX #$00
. 1017  A9 20    LDA #$20
. 1019  9D 00 0C STA $0C00,X
. 101C  9D 00 0D STA $0D00,X
. 101F  9D 00 0E STA $0E00,X
. 1022  9D 00 0F STA $0F00,X
. 1025  A9 00    LDA #$00
. 1027  9D 00 08 STA $0800,X
. 102A  9D 00 09 STA $0900,X
. 102D  9D 00 0A STA $0A00,X
. 1030  9D 00 0B STA $0B00,X
. 1033  CA       DEX
. 1034  D0 E1    BNE $1017
. 1036  A0 12    LDY #$12
. 1038  B9 9F 11 LDA $119F,Y
. 103B  99 82 0C STA $0C82,Y
. 103E  99 2A 0F STA $0F2A,Y
```

```
. 1041  B9 B2 11 LDA $11B2,Y
. 1044  99 D2 0C STA $0CD2,Y
. 1047  99 7A 0F STA $0F7A,Y
. 104A  B9 C5 11 LDA $11C5,Y
. 104D  99 C2 0D STA $0DC2,Y
. 1050  B9 D8 11 LDA $11D8,Y
. 1053  99 62 0E STA $0E62,Y
. 1056  A9 26    LDA #$26
. 1058  99 C2 09 STA $09C2,Y
. 105B  99 62 0A STA $0A62,Y
. 105E  88       DEY
. 105F  10 D7    BPL $1038
. 1061  86 D0    STX $D0
. 1063  86 DB    STX $DB
. 1065  20 8F 10 JSR $108F
. 1068  20 B1 10 JSR $10B1
. 106B  A9 FD    LDA #$FD
. 106D  8D 08 FF STA $FF08
. 1070  AD 08 FF LDA $FF08
. 1073  A2 00    LDX #$00
. 1075  4A       LSR
. 1076  4A       LSR
. 1077  4A       LSR
. 1078  B0 01    BCS $107B
. 107A  CA       DEX
. 107B  4A       LSR
. 107C  B0 01    BCS $107F
. 107E  E8       INX
. 107F  8A       TXA
. 1080  18       CLC
. 1081  65 D0    ADC $D0
. 1083  C9 F7    CMP #$F7
. 1085  F0 E1    BEQ $1068
. 1087  C9 09    CMP #$09
. 1089  F0 DD    BEQ $1068
. 108B  85 D0    STA $D0
. 108D  D0 D9    BNE $1068
. 108F  78       SEI
. 1090  A9 73    LDA #$73
. 1092  8D 14 03 STA $0314
. 1095  A9 11    LDA #$11
. 1097  8D 15 03 STA $0315
```

```
. 109A  A9 02    LDA #$02
. 109C  8D 0A FF STA $FF0A
. 109F  A9 00    LDA #$00
. 10A1  85 DA    STA $DA
. 10A3  85 D9    STA $D9
. 10A5  A9 5A    LDA #$5A
. 10A7  8D 0B FF STA $FF0B
. 10AA  A9 00    LDA #$00
. 10AC  8D 07 FF STA $FF07
. 10AF  58       CLI
. 10B0  60       RTS
. 10B1  A5 DB    LDA $DB
. 10B3  F0 FC    BEQ $10B1
. 10B5  A9 00    LDA #$00
. 10B7  85 DB    STA $DB
. 10B9  A5 D9    LDA $D9
. 10BB  18       CLC
. 10BC  65 D0    ADC $D0
. 10BE  A8       TAY
. 10BF  29 07    AND #$07
. 10C1  85 D9    STA $D9
. 10C3  98       TYA
. 10C4  29 08    AND #$08
. 10C6  D0 01    BNE $10C9
. 10C8  60       RTS
. 10C9  A2 05    LDX #$05
. 10CB  A9 0D    LDA #$0D
. 10CD  85 D2    STA $D2
. 10CF  85 D4    STA $D4
. 10D1  A9 09    LDA #$09
. 10D3  85 D6    STA $D6
. 10D5  85 D8    STA $D8
. 10D7  A9 90    LDA #$90
. 10D9  85 D1    STA $D1
. 10DB  98       TYA
. 10DC  10 31    BPL $110F
. 10DE  A0 FF    LDY #$FF
. 10E0  A5 D1    LDA $D1
. 10E2  18       CLC
. 10E3  69 28    ADC #$28
. 10E5  85 D1    STA $D1
. 10E7  85 D3    STA $D3
```

```
. 10E9  85 D5    STA $D5
. 10EB  85 D7    STA $D7
. 10ED  90 08    BCC $10F7
. 10EF  E6 D2    INC $D2
. 10F1  E6 D4    INC $D4
. 10F3  E6 D6    INC $D6
. 10F5  E6 D8    INC $D8
. 10F7  E6 D3    INC $D3
. 10F9  E6 D7    INC $D7
. 10FB  C8       INY
. 10FC  B1 D3    LDA ($D3),Y
. 10FE  91 D1    STA ($D1),Y
. 1100  B1 D7    LDA ($D7),Y
. 1102  91 D5    STA ($D5),Y
. 1104  C0 27    CPY #$27
. 1106  D0 F3    BNE $10FB
. 1108  CA       DEX
. 1109  D0 D3    BNE $10DE
. 110B  84 DA    STY $DA
. 110D  F0 2D    BEQ $113C
. 110F  A0 26    LDY #$26
. 1111  A5 D1    LDA $D1
. 1113  18       CLC
. 1114  69 28    ADC #$28
. 1116  85 D1    STA $D1
. 1118  85 D3    STA $D3
. 111A  85 D5    STA $D5
. 111C  85 D7    STA $D7
. 111E  90 08    BCC $1128
. 1120  E6 D2    INC $D2
. 1122  E6 D4    INC $D4
. 1124  E6 D6    INC $D6
. 1126  E6 D8    INC $D8
. 1128  E6 D3    INC $D3
. 112A  E6 D7    INC $D7
. 112C  B1 D1    LDA ($D1),Y
. 112E  91 D3    STA ($D3),Y
. 1130  B1 D5    LDA ($D5),Y
. 1132  91 D7    STA ($D7),Y
. 1134  88       DEY
. 1135  10 F5    BPL $112C
. 1137  CA       DEX
```

```
. 1138  D0 D5     BNE $110F
. 113A  86 DA     STX $DA
. 113C  A2 05     LDX #$05
. 113E  A4 DA     LDY $DA
. 1140  A9 0D     LDA #$0D
. 1142  85 D2     STA $D2
. 1144  A9 09     LDA #$09
. 1146  85 D6     STA $D6
. 1148  A9 90     LDA #$90
. 114A  85 D1     STA $D1
. 114C  85 D5     STA $D5
. 114E  A5 D1     LDA $D1
. 1150  18        CLC
. 1151  69 28     ADC #$28
. 1153  85 D1     STA $D1
. 1155  85 D5     STA $D5
. 1157  90 04     BCC $115D
. 1159  E6 D2     INC $D2
. 115B  E6 D6     INC $D6
. 115D  98        TYA
. 115E  49 27     EOR #$27
. 1160  A8        TAY
. 1161  B1 D1     LDA ($D1),Y
. 1163  85 DC     STA $DC
. 1165  B1 D5     LDA ($D5),Y
. 1167  A4 DA     LDY $DA
. 1169  91 D5     STA ($D5),Y
. 116B  A5 DC     LDA $DC
. 116D  91 D1     STA ($D1),Y
. 116F  CA        DEX
. 1170  D0 DC     BNE $114E
. 1172  60        RTS
. 1173  AD 09 FF  LDA $FF09
. 1176  8D 09 FF  STA $FF09
. 1179  AE 0B FF  LDX $FF0B
. 117C  A9 40     LDA #$40
. 117E  E0 83     CPX #$83
. 1180  F0 08     BEQ $118A
. 1182  05 D9     ORA $D9
. 1184  A2 83     LDX #$83
. 1186  86 DB     STX $DB
. 1188  D0 02     BNE $118C
```

```
. 118A  A2 5A    LDX #$5A
. 118C  8E 0B FF STX $FF0B
. 118F  29 07    AND #$07
. 1191  EC 1D FF CPX $FF1D
. 1194  F0 FB    BEQ $1191
. 1196  8D 07 FF STA $FF07
. 1199  68       PLA
. 119A  A8       TAY
. 119B  68       PLA
. 119C  AA       TAX
. 119D  68       PLA
. 119E  40       RTI

>119F 20 04 09 05 13 05 12 20 :R ......
>11A7 14 05 09 0C 20 08 09 05 :R.... ...
>11AF 12 20 20 20 20 20 02 0C :R.     ..
>11B7 05 09 02 14 20 13 14 05 :R.... ...
>11BF 08 05 0E 20 20 20 16 05 :R...   ..
>11C7 12 13 03 08 09 05 02 05 :R........
>11CF 20 04 01 13 20 08 09 05 :R ... ...
>11D7 12 0D 09 14 20 05 09 0E :R.... ...
>11DF 05 0D 20 0A 0F 19 13 14 :R.. .....
>11E7 09 03 0B 21 AA AA AA AA :R...!****
```

2.9 TECHNICAL DETAILS OF TED-CHIP

In all different modes the TED-Chip displays 25 lines of 40 characters. Each character on the screen can accept 16 different colours in 8 different brightness levels. Furthermore the characters can be shown flashing.

The values in the Video-RAM determine which character is to be shown in a certain place. To this belongs a certain attribute which supplies colour, brightness and flashing.

The TED-Chip receives this information from the Video-RAM or Colour-RAM. The Video-RAM consists of 1000 memory places which each consist of an 8 Bit pointer (see also chapter 2.1). The Video Matrix Register in the TED-Chip determines where the Video-RAM is located in the memory plan of the C-16. The registers of the TED-Chip are found in the memory plan of the C-16 from $FF00 (decimal: 65280). The Video Matrix Register is the TED register 20, the complete address is:65280+20=65300 (hex: $FF00+$14 = $4FF14). If we want to change the position of the Video-RAM in the C-16's memory we have to manipulate this register. The TED-Register 20 looks as follows:

Bit:	7	6	5	4	3	2	1	0
Function:	VM4	VM3	VM2	VM1	VM0	NU	NU	NU

NU means "Not Used" and can be ignored, VM4-VM1 determine the position of the Video-RAM in the memory, see the following table:

VM4-VM1	Video-RAM	VM4-VM1	Video-RAM
00000	$0400	10000	$8400
00001	$0C00 (standard)	10001	$8C00
00010	$1400	10010	$9400
00011	$1C00	10011	$9C00
00100	$2400	10100	$A400
00101	$2C00	10101	$AC00
00110	$3400	10110	$B400
00111	$3C00	10111	$BC00
01000	$4400	11000	$C400
01001	$4C00	11001	$CC00
01010	$5400	11010	$D400
01011	$5C00	11011	$DC00
01100	$6400	11100	$E400
01101	$6C00	11101	$EC00
01110	$7400	11110	$F400
01111	$7C00	11111	$FC00

If you possess a C-16 without memory expansion you can in principle put your Video-RAM in 8 different positions ($0400...$3C00). Please make sure that the Colour-RAM is always installed exactly 1K below the Video-RAM. If you (as after switching-on) have the Video-RAM at $0C00, the Colour-RAM will be at $0800.

Each memory place in the Video-Matrix is used as a pointer for the character definition of the corresponding character. The 8th Bit (MSB) of this pointer can be interpreted in 2 different ways. When the RVS-Bit (Bit 7) of TED-Register 7 ($FF07) is 0, the MSB of the Video-Matrix (VM7) is used to decide whether the character should be reverse or normal. If VM7 is 0, it is normal, if it is 1, it is reverse, whereby the TED-Chip will create the reverse character by the hardware . Thus 128 characters can be user defined. If the RVS-Bit of the TED-Chip is 1, 256 own characters can be defined.

VIDEO MATRIX ADDRESS

A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0

VM4 VM3 VM2 VM1 VM0 1 VC9 VC8 VC7 VC6 VC5 VC4 VC3 VC2 VC1 VC0

The Attribute-Memory also consists of 1000 consecutive memory places and contains the Flash-Bit (Bit 7), the brightness (Bit 4-6) and the colour (Bit 0-3). The position of the Attribute- or Colour-RAM is also determined by the Video-Matrix-Register. But since A10 always equals 0, the Attribute-RAM is always 1K below the Video-RAM.

ATTRIBUTE ADDRESS

A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0

VM4 VM3 VM2 VM1 VM0 0 VC9 VC8 VC7 VC6 VC5 VC4 VC3 VC2 VC1 VC0

Each character consists of a matrix of 8x8 pixels which are stored in ROM as 8 consecutive Bytes. The address of this character memory is determined by CB4-CB0 of the TED-Register 19 ($FF13). These Bits are the higher value Bits of the Character-Set-Address.

A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0

CB5 CB4 CB3 CB2 CB1 VM7 VM6 VM5 VM4 VM3 VM2 VM1 VM0

CB0 (if RVS-Bit is 1)

3.1 MUSIC WITH THE BASIC-COMMANDS

The Commodore C-16 has simple but good facilities of sound creation. It has two independent sound channels which can be very easily programmed with the corrosponding BASIC commands VOL (volume) and SOUND (voice, frequency, duration). We don't want to go into too much detail about these BASIC commands (for this you have the manual), but would like to show you with a small example how to programme small music pieces for two voices in BASIC.

```
100 VOL 8
110 DIM N1%(66),N2%(66),D1%(66),D2%(66)
120 I=0
130 READ N1%(I),D1%(I): IF N1%(I)<0 THEN 150
140 I=I+1:GOTO 130
150 T1=I :I=0
160 READ N2%(I),D2%(I): IFN2%(I) <0 THEN 180
170 I=I+1 : GOTO 160
180 I1=-1 :I2=-1
190 IF D1 > 0 THEN 220:ELSE SOUND 1,N1,0
200 I1=I1+1: IF I1 < T1 THEN D1=D1%(I1):N1=N1%(I1):ELSE 270
210 IF N1>0 THEN SOUND 1,N1,300
220 IF D2>0 THEN 250:ELSE SOUND 2,N2,0
230 I2=I2+1 : D2=D2%(I2) : N2=N2%(I2)
240 IF N2>0 THEN SOUND 2,N2,300
250 D1=D1-1:D2=D2-1
260 FOR I=1 TO 80: NEXT I : GOTO 190
270 VOL 0
280 DATA 0,1,685,1,770,1,810,1
290 DATA 798,1,685,1,798,1,834,1
300 DATA 810,2,854,2,755,2,854,2
310 DATA 770,1,685,1,770,1,810,1
320 DATA 798,1,685,1,798,1,834,1
330 DATA 810,2,770,2,0,4
340 DATA 0,1,854,1,810,1,854,1
350 DATA 770,1,810,1,685,1,739,1
360 DATA 704,2,770,2,834,2,864,2
370 DATA 864,1,834,1,798,1,834,1
380 DATA 739,1,798,1,643,1,704,1
```

```
390 DATA 685,2,739,2,810,2,254,2
400 DATA 854,1,810,1,770,1,810,1
410 DATA 704,2,834,2,834,1,798,1
420 DATA 739,1,798,1,685,2,810,2
430 DATA 810,1,770,1,704,1,770,1
440 DATA 643,2,798,2,810,6
450 DATA -1,-1
460 DATA 7,2,516,4,485,2
470 DATA 516,1,345,1,516,1,596,1
480 DATA 571,1,345,1,571,1,643,1
490 DATA 596,2,516,2,485,2,345,2
500 DATA 516,1,345,1,516,1,596,1
510 DATA 571,1,345,1,571,1,643,1
520 DATA 596,2,516,2,596,2,516,2
530 DATA 643,1,516,1,383,1,516,1
540 DATA 262,1,383,,1,7,1,169,1
550 DATA 118,2,262,2,453,2,571,2
560 DATA 571,1,453,1,345,1,453,1
570 DATA 169,1,345,1,118,1,118,1
580 DATA   7,2,169,2,262,2,383,1
590 DATA 118,1,262,1,118,2,118,2
600 DATA 169,1,345,1,7,1,169,1
610 DATA   7,2,7,2
620 DATA 118,1,453,1,383,1,453,1
630 DATA 596,6
640 DATA -1,-1
```

EXPLANATION LINE BY LINE

100 Volume on max.

110 Dimension data fields

120 I counts the number of notes

130 Key note and duration for first voice in; a negative figure means: Ready!

140 Increment counter I and continue.

150 T1 is the total number of notes for voice 1. Put I back to 0.

160 Key in note and duration for voice 2 ; a negative figure means: Ready!

170 Increment counter I and continue.

180 I1 and I2 are pointers for the data fields.

190 When voice 1 is ready, go to 220; otherwise stop voice 1.

200 Increment pointer of voice 1. When ready, leave loop; otherwise put note and duration.

210 If no pause, start note.

220 When voice 2 is ready go to 250; otherwise stop voice 2.

230 Increment pointer of voice 2 and put note and duration.

240 If it is no pause start note.

250 Decrement duration.

260 Wait a short while. In this line you can play all notes shorter or longer.

270 Switch volume off.

280-440 Data for voice 1.

450 End of data for voice 1.

460-630 Data for voice 2.

640 End of data for voice 2.

3.2 THE SOUND REGISTERS OF TED

The Sound Registers are on the same chip as the Graphic Registers in TED. Because of the simple sound possibilities, only 5 registers are required which have the following functions:

ADDRESS	BITS	FUNCTION
$FF0E	0-7	Low Byte of voice 1 frequency
$FF0F	0-7	" " " " 2 "
$FF10	0-1	Top 2 Bits of voice 1 frequency
$FF11	0-3	Volume
	4	Select voice 1 (0=off,1=on)
	5	" " 2 " " " "
	6	" noise for voice 2 (0=off,1=on)
	7	Sound switch (0=on,1=off)
$FF12	0-1	Top 2 Bits of voice 2 frequency

In order to create a sound, choose first the voice or voices and the volume with register $FF11. Normally you set Bit 7 of this register to 1 for no sound. Next you choose the frequencies which you should note that Bits 2-7 of register $FF12 are used for other purposes and must not be changed. Now you can switch on the sound with Bit 7 of $FF11 and or switch it off.

3.3 INTERRUPT PROGRAMMED MUSIC

In this chapter we would like to investigate the possibility of creating music on the C-16 which does not interfere with the rest. This kind of background music is often found in games and it is very practical to concentrate on the game and not on the music. This automatic background music can be created very easily on the C-16. As you perhaps know there is a so-called Interrupt every 1/60 th of a second in the C-16. This is a short interrupt of the normal program during which the C-16 does routine tasks like checking if any key has been pressed. We can use this interrupt so that C-16 besides doing other tasks also actuates the relevant sound parameters. To facilitate things, it is best you key in the following program and see for yourself. There will be a more detailed explanation of this program in the next pages.

```
100 FOR I=1536 TO 1536+173
110 : READ A
120 : POKE I,A
130 : C=C+A
140 NEXT I
150 IF C<>23952 THEN PRINT "ERROR":STOP
160 FOR I=1712 TO 1712+119 STEP 3
170 : READ A,B
180 : POKE I,A : POKE I+1,B-(INT(B/256)*256)
190 : POKE I+2,INT(B/256)
200 : D=D+A+B
210 NEXT I
220 IF D<>34808 THEN PRINT "ERROR":STOP
230 POKE 208,176 : POKE 209,6
240 VOL 7 : SYS 1536
250 :
260 DATA 120,169,36,141,20,3,169,6,141,21,3,169,255,141,252,4
270 DATA 141,254,4,88,96,120,169,14,141,20,3,169,206,141,21,3
280 DATA 88,96,255,0,173,252,4,201,255,240,3,76,14,206,160,0
290 DATA 162,0,177,208,149,210,200,232,138,201,3,208,245,165,208,105
300 DATA 2,133,208,144,2,230,209,165,210,201,255,208,6,32,21,6
310 DATA 24,144,216,201,254,208,19,165,208
315 DATA 133,213,165,211,133,208,165
```

```
320 DATA 209,133,214,165,212,133,209,24,144
325 DATA 196,201,253,208,11,165,213
330 DATA 133,208,165,214,133,209,24,144,181,201,252,208,13,173,17,255
340 DATA 41,240,5,211,141,17,255,24,144,164,73,255,141,252,4,169
350 DATA 255,141,254,4,173,18,255,41,252,5,212,141,18,255,165,211
360 DATA 141,14,255,173,17,255,9,16,141,17,255,24,144,162
370 :
380 DATA 254,1727,254,1796,254,1796,254,1727,254,1712
390 DATA 20,685,20,739,40,739,20,685,20,643,40,596,20,643,20,685
400 DATA 20,739,20,685,80,643,20,685,20,739,40,739
410 DATA 20,685,20,643,40,596,20,643,20,685,20,643
420 DATA 20,596,80,596,253,0
430 DATA 20,770,20,810,40,810,20,796,20,739,40,770
440 DATA 20,770,20,810,20,798,20,739,80,770,253,0
```

MACHINE CODE PROGRAM

```
. 0600 78       SEI
. 0601 A9 24    LDA #$24
. 0603 8D 14 03 STA $0314
. 0606 A9 06    LDA #$06
. 0608 8D 15 03 STA $0315
. 060B A9 FF    LDA #$FF
. 060D 8D FC 04 STA $04FC
. 0610 8D FE 04 STA $04FE
. 0613 58       CLI
. 0614 60       RTS
. 0615 78       SEI
. 0616 A9 0E    LDA #$0E
. 0618 8D 14 03 STA $0314
. 061B A9 CE    LDA #$CE
. 061D 8D 15 03 STA $0315
. 0620 58       CLI
. 0621 60       RTS
. 0622 EA       NOP
. 0623 EA       NOP
. 0624 AD FC 04 LDA $04FC
. 0627 C9 FF    CMP #$FF
. 0629 F0 03    BEQ $062E
. 062B 4C 0E CE JMP $CE0E
. 062E A0 00    LDY #$00
```

```
. 0630 A2 00    LDX #$00
. 0632 B1 D0    LDA ($D0),Y
. 0634 95 D2    STA $D2,X
. 0636 C8       INY
. 0637 E8       INX
. 0638 8A       TXA
. 0639 C9 03    CMP #$03
. 063B D0 F5    BNE $0632
. 063D A5 D0    LDA $D0
. 063F 69 02    ADC #$02
. 0641 85 D0    STA $D0
. 0643 90 02    BCC $0647
. 0645 E6 D1    INC $D1
. 0647 A5 D2    LDA $D2
. 0649 C9 FF    CMP #$FF
. 064B D0 06    BNE $0653
. 064D 20 15 06 JSR $0615
. 0650 18       CLC
. 0651 90 D8    BCC $062B
. 0653 C9 FE    CMP #$FE
. 0655 D0 13    BNE $066A
. 0657 A5 D0    LDA $D0
. 0659 85 D5    STA $D5
. 065B A5 D3    LDA $D3
. 065D 85 D0    STA $D0
. 065F A5 D1    LDA $D1
. 0661 85 D6    STA $D6
. 0663 A5 D4    LDA $D4
. 0665 85 D1    STA $D1
. 0667 18       CLC
. 0668 90 C4    BCC $062E
. 066A C9 FD    CMP #$FD
. 066C D0 0B    BNE $0679
. 066E A5 D5    LDA $D5
. 0670 85 D0    STA $D0
. 0672 A5 D6    LDA $D6
. 0674 85 D1    STA $D1
. 0676 18       CLC
. 0677 90 B5    BCC $062E
. 0679 C9 FC    CMP #$FC
. 067B D0 0D    BNE $068A
. 067D AD 11 FF LDA $FF11
```

```
. 0680 29 F0    AND #$F0
. 0682 05 D3    ORA $D3
. 0684 8D 11 FF STA $FF11
. 0687 18       CLC
. 0688 90 A4    BCC $062E
. 068A 49 FF    EOR #$FF
. 068C 8D FC 04 STA $04FC
. 068F A9 FF    LDA #$FF
. 0691 8D FE 04 STA $04FE
. 0694 AD 12 FF LDA $FF12
. 0697 29 FC    AND #$FC
. 0699 05 D4    ORA $D4
. 069B 8D 12 FF STA $FF12
. 069E A5 D3    LDA $D3
. 06A0 8D 0E FF STA $FF0E
. 06A3 AD 11 FF LDA $FF11
. 06A6 09 10    ORA #$10
. 06A8 8D 11 FF STA $FF11
. 06AB 18       CLC
. 06AC 90 A2    BCC $0650
```

Our interrupt-player's syntax is relatively simple: The volume level and length are stored. Additionally some Bytes have special functions:

$FC (252) Volume. The volume within a piece can be changed. The Low-Byte of the frequency contains the new volume 0 to 8. The High-Byte is not used.

$FD (253) This is a kind of GOSUB-command. It branches off to the address which is given instead of the frequency. Furthermore the old address is stored to allow a return of the sub-program.

$FE (254) This is the RETURN-command which finishes a sub program. The two frequency Bytes have to be there but are not used.

$FF (255) End of sequence.

I

4.1 INTRODUCTION TO MACHINE CODE

If you want to really master your computer (who want's it to be the other way round), you cannot avoid programming in computer language (MACHINE CODE). The nice thing is actually the tremendous speed and unlimited control over the hardware possibilities of the computer.

What often frightens the BASIC-Programmer is the alleged complexity of Machine Code and the fact that when you make a mistake, the computer can crash. This, however, never causes lasting damage. Even an absolute beginner can write small routines in machine code with the integrated machine code monitor within a short period of time. This introduction should, with a few examples help to inspire some experimenting.

WHAT IS MACHINE CODE

The heart of C 16 is a Micro-Processor (7501) which can follow a series of commands. These commands are similar to those of a programmable calculator (TI, HP, etc.). The programmer has the following at his disposal: a calculating register (accumulator), 2 Index-Registers (X-Register and Y-Register) and the 16384 bytes of memory in the C-16. Many memory places of the C-16 have a certain function which you have to know as a Machine Code Programmer. Therefore the knowledge of a Memory Map is indispensable (see chapter 4.4). The memory places in the C-16 can only accept values between 0 and 255 (try from BASIC: POKE 82,256 or similar). A no. in a memory space can be a machine Code command which is understood by the processor and carried out, or some other data; this has to be decided by the user. You don't have to remember which figure denotes which machine code command, this is done by the integral Mini-Assembler in the built-in Machine CODE Monitor of the C-16. You only have to remember the so called Mnemonics, which are memory aids with 3 letters for the individual computer commands. From this the Assembler generates the values between 0 and 255 and stores them in the memory.

4.2 EXAMPLES OF THE MOST IMPORTANT COMMANDS

1. The commands LDA and STA, the first program

Let's assume the C-16's frame colour is to be changed to black.In BASIC you could either use the COLOUR-Command or transfer the value directly into the background register of the Video-Chip with a POKE-Command. The BASIC-Line would be:

```
1 COLOR 4,1,0
```

or:

```
1 POKE 65305,0
```

Let's now programme the last line in machine language instead of BASIC. First you have to know where the machine program should run or stored (e.g. called with: SYS 818). The Cassette-Buffer (at $0332-03F2 or 818-1010) is suitable for our little demonstrations because it is only used during a cassette operation.

If you want to poke 0 into memory space 65305 you have to obtain a 0 first. For this you can use the command LDA which means load accumulator. The value, in this case 0 with which the computer register is to be loaded is put directly after the command in the memory. In machine language this is shown by a # sign followed by the required figure. To inform the Mini-Assembler that it is a hexa-decimal number (it cannot use any others), you have to put a $ in front of every number and it has to be a two figure number. The complete command now reads: LDA #$00.Now you have the value in the accumulator and have to store it in memory location 65305 (hex. $FF19). To do this, command STA can be used (STore Accumulator). The required address has to follow, i.e. STA $FF19. Finally you have to go back to BASIC with the command RTS (ReTurn Subroutine):RTS.

In order to programme this small routine (and the following demos) into the C-16, type the BASIC command: MONITOR. When you enter a machine code proramme, you must always use CAPITAL letters only and end each line with the RETURN key.

```
MONITOR  (RETURN)
```

The screen will display.

```
MONITOR

   PC   SR   AC   XR   YR   SP

; 3AB3  00   00   00   00   F9
```

Now Type

```
A 0332  LDA #$00              (RETURN)

A 0334  STA $FF19             (RETURN)

A 0337  RTS                   (RETURN)

A 0338                        (RETURN)

X                             (RETURN)

READY.
```

Since our program ends with the RTS-command we can leave the Mini-Assembler. When it shows: A 0338, simply press RETURN and leave the machine code monitor by typing X (RETURN).

You can now start your first machine code program with the command: SYS 818. If you haven't given up yet you can look at the programming of a loop in the next chapter.

2. A PERMANANT LOOP (INC, JMP)

If you wanted to change the frame colour continually, you would have to write the following program in BASIC:

```
1 FOR I=0 TO 127

2 : POKE 65305,I

3 NEXT I

4 GOTO 1
```

In order to change the frame colour, you can always increment the content of memory space 65305 by one. Use command: INC address (INCrement). This command increments the contents of address by one. If the memory space is full (>255), it automatically reverts to zero again. Our first command is: INC $FF19. Since we want to increment the content of $FF19 always by one we have to jump back to this command. For this, we can use the command JMP address (JuMP). As our program is to start at $0332, the second (and last) line reads: JMP $0332. Once again, type MONITOR to enter monitor and type the following:

```
A 0332 INC $FF19

A 0335 JMP $0332
```

With: SYS 818 you will see the first effect which cannot be realised in BASIC.

Since there is no program end, so you cannot return with RTS into BASIC and will have to use the RESET key.

3. DECEISION LOOP (LDX,DEX,BNE,NOP)

The example above is now to be extended by a time loop. In order to programme a time loop, we can for example

1. Set a counter to certain no.
2. Decrement counter by one
3. Check if the counter is 0
4. If not, repeat the process until 0.

The time taken by computer to do this is a time loop and is very short. The program first must have a counter and secondly make a deceision if the counter is zero.

For this purpose one of the two Index-Registers (X or Y) come in useful. The content can be between 0 and 255 (hex. $FF).

Our program is to start again with the command: INC $FF19, i.e. it is to increment the frame colour by one. Next, the Index-Register X, which we want to decrement is loaded with the start value similar to the LDA-command with: LDX address; in our case: LDX #$B1, where LDX means LoaD X register. Decrementing the X-Register by one is simple and is done with the command DEX (DEcrement Xregister). After decrementing we have to check whether the X-Register has reached 0. For this we use the command: BNE (Branch Not Equal). BNE tests whether the previous machine command (DEX) has set a certain Flag, the ZERO-Flag. This flag is set (=1) when X reg. contains a 0 after DEX operation. If not, the program jumps back to the address after BNE: BNE $0337 i.e. it jumps back to the command DEX and forms a loop.

In order to wait a short while the command NOP (No OPeration) can be utilised. At the end of our program we want to start from the beginning, i.e. JMP $0332. The complete program looks now as follows:

```
A 0332 INC $FF19

A 0335 LDX #$B1

A 0337 DEX
```

```
A 0338 BNE $0337

A 033A NOP

A 033B JMP $0332
```

Try the program with: SYS 818.

Instead of the X-Register, the Y-Register can also be used. Instead of LDX, use LDY (LoaD Yregister) and DEY (DEcrement Yregister) instead of DEX.

4. USING THE OPERATING SYSTEM (JSR)

If the machine program is to talk to the outside world e.g. operate printer or show information on the screen, the easiest way to do this is to use the existing routines in the operating system of the C-16. A summary and an operating instruction of these KERNAL-Routines can be found in chapter 4.4 of this book. As an example, erase the machine code from the screen. In BASIC this would look as follows:

```
1 PRINT CHR$(147)
```

This routine, which brings a character from the current cursor position to the screen, begins in the operating system at address $FFD2 (hex.) and requires the ASCII-Value of the character to be printed in the accumulator. We obtain the character with the familiar LDA command for the accumulator and call the routine with the command JSR (Jump SubRoutine). The Kernel sub routine has all the instructions needed to print the character in the accumulator on the screen. You can therefore effectively use Kernel routines to do mundane tasks like adding, subtracting etc.. The program for erasing the screen in machine code is.

```
A 0332 LDA #$93

A 0334 JSR $FFD2

A 0337 RTS
```

With SYS 818 you can now erase the screen.

5. PRINT MESSAGE (TABLE ADDRESSING)

If you want to print more than one character on the screen it is best to programme a loop which refers to a table. The next character will be obtained automatically from the table. The end of the table should contain a zero, so that your program knows where to finish the loop.

The table can accessed by the command LDA address,X. This command loads the accumulator with the CONTENTS of memory address indexed by X register. The value in the X-Register plus the value of the address equals the memory address from which the contents are loaded into the accumulator- e.g. LDA $0341,X. If X=2, then the acc. will be loaded with contents of mem. addr. $0343. If a zero (the self-defined end of the table) is loaded into the accumulator, another Flag (the ZERO-Flag) is set which can be tested with the command: BEQ address (Branch if EQal). The value in the X-Register can be changed with DEX or INX (INcrement Xregister) in order to be able to call the next address with another LDA address,X.

```
A 0332 LDX #$00

A 0334 LDA $0341,X

A 0337 BEQ $0340

A 0339 JSR $FFD2

A 033C INX
```

```
A 033D JMP $0334

A 0340 RTS
```

The above routine loaded the acc. with the contents of mem. addr. $0341, which in our case is letter H. As it is not 0, it jumped to sub routine at $FFD2. The X reg. is incremented by 1 and loops back to get the contents of address $0341+1 i.e. $0342. The contents are $41(E). It carries on like this until it loads the contents of mem. addr. $0347 and finds it to be 0. It sets a flag, and the routine jumps to RTS.

To enable us to key in the table with the required message we type into the monitor the command: M 0341, which has the effect of a hexa decimal print out of the memory location from $0341 onwards. We move with the cursor to the first memory location and key in the hexa decimal ASCII values of our message. Our message is to be HELLO, the corresponding ASCII values are: 48 41 4C 4C 4F. After you have typed these figures (don't forget to press RETURN) the screen should look as follows (your input in bold type):

```
M0341

>0341 48 41 4C 4C 4F 00 00 00 :HELLO
```

Leave the machine code monitor with: X (RETURN), erase screen with the CLEAR-key and test your program with: SYS 818.

6. INDIRCT ADDRESSING

In the following example, we want to fill the whole screen with hearts. In BASIC the program would look as follows:

```
1 FOR I=3072 TO 4071

2 : POKE I,83
```

```
3 NEXT I
```

Watch the speed and compare later to our corresponding M/C program. Writing on a larger area in machine code is best done with indirect addressing; the use of a certain memory locations is carried out with the commands:

```
LDA (pointer),Y
```

```
STA (pointer),Y
```

The content of addressed pointer plus the content of the Y-Register equals the address from which a figure is loaded into the accumulator (LDA) or where it will be stored (STA).

The "Pointer" consists of two consecutive memory locations, each address is less than 256. As you know the contents of a single memory location can only accept values between 0 and 255. The computer has, however, 65535 possible memory locations. Therefore the required address has to be divided into two parts. The lower one is stored in the first of the two memory locations, the higher one in the following. After that the command can be used, see the following program:

The value $53 is stored into video RAM (screen) whose start address is $0C00 + Y. Y is inc. by 1 until Y=256($FF) and $53 is stored from $0C00 to $0CFF. The Y reg. becomes 0 as 1 is added to $FF and the prog jumps to RTS.

```
A 0332 LDY #$00

A 0334 LDA #$00

A 0336 STA $D0

A 0338 LDA #$0C

A 033A STA $D1

A 033C LDA #$53
```

```
A 033E STA ($D0),Y

A 0340 INY

A 0341 BNE $033E

A 0343 RTS
```

After call-up with SYS 818, 256 hearts will be printed on the screen. As only a block of 256 can be written on, in order to write the next block, you have to increment the upper byte of pointer by one and repeat the process. Since the screen memory is 1K long and 1K consists of 4 blocks, the above loop has to be repeated 4 times. The X-Register can be used as counter.

Change the above program as follows:

```
A 033E LDX #$04

A 0340 STA ($D0),Y

A 0342 INY

A 0343 BNE $0340

A 0345 INC $D1

A 0347 DEX

A 0348 BNE $0340

A 034A RTS
```

The four blocks are - $0C00-$0CFF, $0D00-$0DFF, $0E00-$0EFF, $0F00-$0FE7. INC $D1 increments the contents of $D0 by 1 i.e. from $0C to $0D. The X reg. acts as a loop counter.

Make sure that the program was programmed correctly (with monitor command: D 0332 034A). Return to BASIC and try it with SYS 818. If

everything is o.k. the speed will surely convince you of the speed of M/C programming. For comparison: our BASIC program requires for this task approx. 5.9 seconds, the M/C program approx. 16 Milli-seconds (=16/1000 seconds); i.e. it is in this case 370 times faster than the corresponding BASIC-program!

7. SUB-ROUTINES (JSR,RTS)

You can write small sub routines to do mundane tasks i.e. move a sprite X a no. of pixels. You can then call this routine from main prog. whenever you need it. The sub routine must have RTS, so that the prog. returns to the next instruction in the main prog.. The command is JSR address. This was already used for printing a message on the screen. The return from this routine is done with the command RTS (ReTurn from Subroutine). This does not jump back into BASIC but to the next address after JSR address.

Erasing the screen can also look as follows:

```
A 0332 JSR $0336

A 0335 RTS

A 0336 LDA #$93

A 0338 JSR $FFD2

A 033B RTS
```

You can now test your first sub-routine with the familiar SYS 818.

8. COMPARISON COMMANDS (CMP,CPX,BCC,BCS)

Often the content of certain memory addresses has to be compared with constants or the contents of other memory locations. This is automatically done with the ZERO Flag when comparing to ZERO (see

paragraph 3). When comparing something else you have to use one of the comparison commands CMP (CoMPare accu.), CPY (ComPare Yregister) or CPX (ComPare Xregister).

After the command it needs to know, what to compare it with. With the # sign, the contents of the corresponding processor register are compared with the indicated number. Without the # sign, with the contents of the addressed memory location.

If the contents of the accumulator are greater or equal to the indicated number or the contents of the addressed mem. location, a certain flag, the CARRY Flag, is set to 1. On equality the ZERO Flag is set to 1 and on unequality to 0.

If the contents of the accumulator are less, the CARRY-Flag is set to 0.

	CARRY	ZERO
Acc.> or = Ind. No.	1	0
Acc. = Ind. No.	0	1
Acc.< or = Ind. No.	0	0

The information in the Flags can now be used by the branching commands:

BNE Branch if Not Equal ---Zero Flag

BEQ Branch if Equal ------- " "

BCC Branch if Carry Clear ---Carry Flag

BCS Branch if Carry Set ----- " "

Here is an example to test branching:

```
A 0332 LDA #$F1

A 0334 CMP $FF19
```

```
A 0337 BCC $033C

A 0339 INC $FF19

A 033C RTS
```

As long as the acc. is less than or equal to the contents of $FF19, CARRY flag is 0, the program loops back changing the frame colour.

A further example of a comparison test:

```
A 0332 LDX $0C00

A 0335 CPX #$01

A 0337 BEQ $0340

A 0339 BCS $0341

A 033B LDA #$00

A 033D STA $FF19

A 0340 RTS

A 0341 LDA #$F1

A 0343 STA $FF19

A 0346 RTS
```

If an A is shown in the top left screen corner, the program doesn't do anything, it jumps immediately back into BASIC. Characters with a larger code switch the frame to white, the smaller ones (@) to black.

Anybody who has reached this stage and tried all examples and even modified a bit can be sure to enjoy the Assembler-Programming. For

more thorough introduction to Machine Code programming, you should study some of many books published on the subject. We can recommend the book "Programming the 6502" by Rodney Zaks, published by SYBEX publishers (the 6502 has exactly the same command set as the 7501 in the C-16), as well as many other books which deal with the 6502. The next chapter will give you a good idea of all the commands of the 6502/7501. And now: keep up your courage!!!

A Few More Tips For Beginners

Machine language is best learnt by studying other machine language programs. Such programs are published in magazines. Familiarise yourself with the program even if it refers to a different computer but uses the Micro-Processor 6502 (or 6510). Make sure, you understand the code. This requires persistence, specially if it deals with a technique unknown to you.This can prove extremely tiresome but with enough patience you will be the winner.

Once you have looked at other machine language programs, you have to start writing some of your own. They can be utility programs for your BASIC-programs or a pure machine language program. Remember : Practice makes perfect!

You should also use the routines available in the computer in a program which helps you to write, prepare and check machine programs. An example is KERNAL (see chapter 4.4) which allows keyboard scan, text display, control of peripherals such as cassette player, disc drives, printer, etc.. KERNAL is extremely powerful and its usage is highly recommended.

The integral Micro Processor 7501 in the C-16 is a further development of series 6502 (Atari, Apple, PET, VC-20) and 6510 (C-64). It is therefore fully software compatible to them and possesses exactly the same command set. This facilitates the re-writing of programs of these computers tremendously. The only difference is the different programming of input/output (Floppy, Cassette), graphics or sound. If you have a program for the Commodore C-64 you can consult our comparison table in chapter 4.6.

For all beginners, as an introduction and for all advanced programmers for consultation, we now list alphabetically all commands of the 7501 (6502/6510 resp.). A detailed description of each individual command would be too elaborate for this book. For this purpose there are already enough books in the bookshops.

The list shows the command and a short description, the way of addressing, the assembler language format, the OPC-Code, the no. of Bytes and clock cycles (+ signifies that, depending on memory pages, 1 or 2 more cycles are used).

COMMAND	Description	Addressing	Assembler Format	Op Code	No.Of Bytes	No.Of Cycles
ADC	Add To Acc. With Carry					
		Immediate	ADC #Operand	69	2	2
		Zero Page	ADC Operand	65	2	3
		Zero Page,X	ADC Operand,X	75	2	4
		Absolute	ADC Operand	6D	3	4
		Absolute,X	ADC Operand,X	7D	3	4+
		Absolute,Y	ADC Operand,Y	79	3	4+
		(Indirect,X)	ADC (Operand,X)	61	2	6
		(Indirect,Y)	ADC (Operand,Y)	71	2	5+

AND	Logical AND To Accumulator					
	Immediate	AND	#Operand	29	2	2
	Zero Page	AND	Operand	25	2	3
	Zero Page,X	AND	Operand,X	35	2	4
	Absolute	AND	Operand	2D	3	4
	Absolute,X	AND	Operand,X	3D	3	4+
	Absolute,Y	AND	Operand,Y	39	3	4+
	(Indirect,X)	AND	(Operand,X)	21	2	6
	(Indirect,Y)	AND	(Operand,Y)	31	2	5+
ASL	Accumulator Shift Left					
	Accumulator	ASL	A	0A	1	2
	Zero Page	ASL	Operand	06	2	5
	Zero Page,X	ASL	Operand,X	16	2	6
	Absolute	ASL	Operand	0E	3	6
	Absolute,X	ASL	Operand,X	1E	3	7
BCC	Branch On Carry Clear					
	Relative	BCC	Operand	90	2	2+
BCS	Branch On Carry Set					
	Relative	BCS	Operand	B0	2	2+
BEQ	Branch On Result = 0					
	Relative	BEQ	Operand	F0	2	2+
BIT	Test Mem. Bits					
	Zero Page	BIT	Operand	24	2	3
	Absolute	BIT	Operand	2C	3	4
BMI	Branch On Result Minus					
	Relative	BMI	Operand	30	2	2+

BNI Branch On Result Not Equal to 0	Relative	BNE	Operad	D0	2	2+
BPL Branch On Result Plus	Relative	BPL	Operand	10	2	2+
BRK Force Break	Implied	BRK	Operand	00	1	7
BVC Branch On Overflow Clear	Relative	BVC	Operand	50	2	2+
CLC Clear Carry Flag	Implied	CLC		18	1	2
CLD Clear Decimal Mode	Implied	CLD		D8	1	2
CLI Clear IRQ Int. Disable Bit	Implied	CLI		58	1	2
CLV Clear Overflow Flag	Implied	CLV		B8	1	2
CMP Compare Mem. And Acc.	Immediate	CMP	#Operand	C9	2	2
	Zero Page	CMP	Operand	C5	2	3
	Zero Page,X	CMP	Operand,X	D5	2	2
	Absolute	CMP	Operand	CD	3	4
	Absolute,X	CMP	Operand,X	DD	3	4+
	Absolute,Y	CMP	Operand,Y	D9	3	4+
	(Indirect,X)	CMP	(Operand,X)	C1	2	6+
	(Indirect,Y)	CMP	(Operand,Y)	D1	2	5+

CPX Compare Mem. And Index X

Immediate	CPX #Operand	E0	2	2
Zero Page	CPX Operand	E4	2	3
Absolute	CPX Operand	EC	3	4

CPY Compare Mem. And Index Y

Immediate	CPY #Operand	C0	2	2
Zero Page	CPY Operand	C4	2	3
Absolute	CPY Operand	CC	3	4

DEC Decrement Mem. By One

Zero Page	DEC Operand	C6	2	5
Zero Page,X	DEC Operand,X	D6	2	6
Absolute	DEC Operand	CE	3	6
Absolute,X	DEC Operand,X	DE	3	7

DEX Decrement Index X By 1

Implied	DEX	CA	1	2

DEY Decrement Index Y By 1

Implied	DEY	88	1	2

EOR Exclusive OR Mem. With Acc.

Immediate	EOR #Operand	49	2	2
Zero Page	EOR Operand	45	2	3
Zero Page,X	EOR Operand,X	55	2	4
Absolute	EOR Operand	4D	3	4
Absolute,X	EOR Operand	5D	3	4+
Absolute,Y	EOR Operand	59	3	4+
(Indirect,X)	EOR (Operand),X	41	2	6
(Indirect),Y	EOR (Operand),Y	51	2	5+

INC Increment Mem. By One

Zero Page	INC Operand	E6	2	5
Zero Page,X	INC Operand,X	F6	2	6
Absolute	INC Operand	EE	3	6
Absolute,X	INC Operand,X	FE	3	7

INX Increment Index X By One

Implied	INX	E8	1	2

INY Increment Index Y By One

Implied	INY	C8	1	2

JMP Jump To An Address

Absolute	JMP Operand	4C	3	3
Indirect	JMP (Operand)	6C	3	5

JSR Jump To Sub-Routine

Absolute	JSR Operand	20	3	6

LDA Load Acc. With Mem.

Immediate	LDA #Operand	A9	2	2
Zero Page	LDA Operand	A5	2	3
Zero Page,X	LDA Operand,X	B5	2	4
Absolute	LDA Operand	AD	3	4
Absolute,X	LDA Operand,X	BD	3	4+
Absolute,Y	LDA Operand,Y	B9	3	4+
(Indirect,X)	LDA (Operand,X)	A1	2	6
(Indirect),Y	LDA (Operand),Y	B1	2	5+

LDX Load X Reg.

Immediate	LDX #Operand	A2	2	2
Zero Page	LDX Operand	A6	2	3
Zero Page,Y	LDX Operand,Y	B6	2	4
Absolute	LDX Operand	AE	3	4
Absolute,Y	LDX Operand,Y	BE	3	4+

LDY	Load Y Reg.					
	Immdiate	LDY	#Operand	A0	2	2
	Zero Page	LDY	Operand	A4	2	3
	Zero Page,X	LDY	Operand,X	B4	2	4
	Absolute	LDY	Operand	AC	3	4
	Absolute,X	LDY	Operand,X	BC	3	4+
LSR	Shift 1 Bit Right					
	Accumulator	LSR	A	4A	1	2
	Zero Page	LSR	Operand	46	2	5
	Zero Page,X	LSR	Operand,X	56	2	6
	Absolute	LSR	Operand	4E	3	6
	Absolute,X	LSR	Operand,X	5E	3	7
NOP	No Operation					
	Implied	NOP		EA	1	2
ORA	Logical OR Acc. With Mem.					
	Immediate	ORA	#Operand	09	2	2
	Zero Page	ORA	Operand	05	2	3
	Zero Page,X	ORA	Operand,X	15	2	4
	Absolute	ORA	Operand	0D	3	4
	Absolute,X	ORA	Operand,X	1D	3	4+
	Absolute,Y	ORA	Operand,Y	19	3	4+
	(Indirect,X)	ORA	(Operand,X)	01	2	6
	(Indirect),Y	ORA	(Operand),Y	11	2	5+
PHA	Push (Store) Acc. In Stack					
	Immediate	PHA		48	1	3
PHP	Store Status Word in Stack					
	Immediate	PHP		08	1	3
PLA	Get Acc. From Stack					
		PLA		68	1	4

PLP	Get Prog. Sta. Word From Stack		PLP		28	1	4
ROL	Rotate Left						
		Accumulator	ROL	A	2A	1	2
		Zero Page	ROL	Operand	26	2	5
		Zero Page,X	ROL	Operand,X	36	2	6
		Absolute	ROL	Operand	2E	3	6
		Absolute,X	ROL	Operand,X	3E	3	7
ROR	Rotate Right						
		Accumulator	ROR	A	6A	1	2
		Zero Page	ROR	Operand	66	2	5
		Zero Page,X	ROR	Operand,X	76	2	6
		Absolute	ROR	Operand	6E	3	6
		Absolute,X	ROR	Operand,X	7E	3	7
RTI	Return From Interrupt						
		Implied	RTI		40	1	6
RTS	Return From Sub-Routine						
		Implied	RTS		40	1	6
SBC	Subtract From Acc. With Carry						
		Immediate	SBC	#Operand	E9	2	2
		Zero Page	SBC	Operand	E5	2	3
		Zero Page,X	SBC	Operand,X	F5	2	4
		Absolute	SBC	Operand	ED	3	4
		Absolute,X	SBC	Operand,X	FD	3	4+
		Absolute,Y	SBC	Operand,Y	F9	3	4+
		(Indirect,X)	SBC	(Operand,X)	E1	2	6
		(Indirect),Y	SBC	(Operand),Y	F1	2	5+

SEC	Set Carry Flag					
		Implied	SEC	38	1	2
SED	Set Decimal Mode					
		Implied	SED	F8	1	2
SEI	Set IRQ Disable Bits					
		Implied	SEI	78	1	2
STA	Store Acc. In Memory					
		Zero Page	STA Operand	85	2	3
		Zero Page,X	STA Operand,X	95	2	4
		Absolute	STA Operand	8D	3	4
		Absolute,X	STA Operand,X	9D	3	5
		Absolute,Y	STA Operand,Y	99	3	5
		(Indirect,X)	STA (Operand,X)	81	2	6
		(Indirect),Y	STA (Operand),Y	91	2	6
STX	Store X Reg. In Memory					
		Zero Page	STX Operand	86	2	3
		Zero Page,Y	STX Operand,Y	96	2	4
		Absolute	STX Operand	8E	3	4
STY	Store Y Reg. In Memory					
		Zero Page	STY Operand	84	2	3
		Zero Page,X	STY Operand,X	94	2	4
		Absolute	STY Operand	8C	3	4
TAX	Transfer Acc. To X Reg.					
		Implied	TAX	AA	1	2
TAY	Transfer Acc. To Y Reg.					
		Implied	TAY	A8	1	2

TSX Transfer Stack Pointer To X REG	Implied	TSX	BA	1	2
TXA Transfer X Reg. To Acc.	Implied	TXA	8A	1	2
TXS Transfer X Reg. To Stack Pointer	Implied	TXS	9a	1	2
TYA Transfer Y Reg. To Acc.	Implied	TYA	98	1	2

4.4 THE USE OF KERNAL-ROUTINES

There is one question which worries all programmers : What to do if the manufacturer changes the computer operating system? Lengthy machine language programs possibly won't function properly without extensive modifications. In order to reduce this problem, Commodore has developed a principle to ease the programmer's lot. This is the so called KERNAL. Essentially KERNAL is a standard jump table for input, output and memory management programs in the operating system. If the system program is improved or modified, the location of the individual programs in ROM can change, but the KERNAL jump table is also altered accordingly.

KERNAL is the operating system of the Commodore 16. All inputs, outputs and the memory controls are controlled by the KERNAL.
There are 39 input/output routines and other very useful utilities which can be accessed via the table. Using these, you not only save time but can also adapt your programs from one Commodore computer to the other. In the C-16, the jump table is on the last page of the memory in ROM..... ($FF49-FFF5).

To use the KERNAL jump table, first you set up th parameters that the KERNAL routines needs to work. Then you jump to the suitable position in the KERNAL jump table using JSR instruction. After finishing the routine, KERNAL transfers the control to your m/c program.
Depending on the KERNAL-Routine used, parameters are returned to your program in certain registers. You will find the relevant registers for the individual KERNAL routine in the description of the KERNAL subroutine.

KERNAL FUNCTIONS ON POWER UP OR RESET

1) After switching on the power the Stack Pointer is initially set to 0, and the decimal mode is erased.

2) Then KERNAL checks if a ROM with auto start (e.g. a game

cartridge) is available at address $8000 (dec. 32768). If it is, the normal initialisation is interrupted and the control transferred to the program stored in ROM. If not, then the normal system initialisation is continued.

3) Next KERNAL initialises all input/output facilities. The serial bus is initialised, the cassette motor switched off and the TED Chip is loaded with the default values.

4) Now KERNAL checks, if the STOP key is pressed. If yes, it branches to the machine language monitor.

5) As the next step KERNAL carries out a RAM test and sets the RAM Pointers at the top and the bottom. Also the Zero Page is initialised and the cassette buffer prepared.

6) Finally KERNAL carries out the following routines: The I/O Vectors are set to default values. The screen erased and all screen editor variables set to zero. Then the BASIC Interpreter is started at $8000.

WORKING WITH KERNAL

When writing programs in machine code, it is often convenient to use operating system routines. These consist of input/output, access to the system clock, memory management and similar functions. It is a waste of time to write them yourself as it is so easy to access the operating system. It helps to speed up the programming too.

In order to work with a KERNAL routines, first you have to make all the necessary preparations. If, for example, a routine initially requires the call up of another KERNAL routine, then you have to call this first. If the routine requires a number in the accumulator, then this number must be there. If the conditions are not fulfilled then

the routines cannot work properly.

After carrying out all preparations you call the routine via the JSR command. All accesible KERNAL routines are structured sub routines with RTS. When the KERNAL routine has finished its task, the control is returned to your program at the instruction after the JSR.

Many KERNAL routines return error codes in the status word or accumulator when faults occur. If you want to program well and create acceptable machine programs these fault indicators must not be ignored.

That's all there is in using the KERNAL. THREE steps.

1) Set Up

2) Call Routine

3) Error Handling

The following conventions are used for the description of KERNAL routines:

NAME: Name of KERNAL routine.

PURPOSE: Short description of purpose of the KERNAL routine.

ADDRESS: This is the call address of the KERNAL routine in hexa decimal.

TRANSFER: Registers under this heading are used for transferring parameters to and from KERNAL routines.

PREPARATION: Certain KERNAL routines require a data to be setup before they can operate. They may also require calling of other KERNAL routines beforehand.

ERRORS: A return from KERNAL routine with the CARRY Flag set indicates that an error was encountered. The accumulator contains the error no..

STACK REQUIREMENT: This is the actual no of stack bytes used by the KERNAL routine.

REGISTER: All registers used by the KERNAL routine are listed here.

DESCRIPTION: Here you will find a short description of the functions of the KERNAL routine.

PROCEDURE: How to use the routine.

EXAMPLE: Illustrates operation with a small example.

NAME: ACPTR

PURPOSE: Get data from the serial bus
ADDRESS: $FFA5
TRANSFER: A
PREPARATION: TALK, TKSA
ERRORS: See READST
STACK REQUIREMENT: 13
REGISTER: A, X

Description:

This routine is used to receive data from a device on the serial bus, e.g. a disc drive. This routine gets a data byte directly from the serial bus. This data is transferred to the accumulator. As preparation, the TALK routine has to be called to command the device on the serial bus to send data on bus. If the input device needs a secondary command,it must be sent using the KERNAL routine TKSA before calling this routine. ERRORS are returned in the status word. Use READST routine to read the status word.

PROCEDURE:

1) Command the device on the serial bus to prepare to send data to the C-16. (Use KERNAL routines TALK and TKSA.)

2) Call this routine (via JSR).

3) Use or store data.

EXAMPLE:

```
$1010                              ;Fetch a byte from bus

$1010      20 A5 FF      JSR ACPTR

$1013      85 D0         STA DATA
```

NAME: CHKIN

PURPOSE: Open a channel for input

ADDRESS: $FFC6

TRANSFER: X

PREPARATION: (OPEN)

ERRORS:

STACK REQUIREMENT: None

REGISTER: A, X

Description:

Each logical file that has already been opened by the KERNAL routine OPEN can be defined as an input channel by this routine. Naturally the device has to be an input device otherwise an error will occur and the routine will abort.

If the data is not via the keyboard, this routine has to be called before usin CHRIN or GETIN. If the input is via the keyboard and no other input channels are open, then this routine and the OPEN routine are not required.

If this routine is used with a device on the serial bus, it automatically sends the talk address via the bus and the secondary address, if one was specified by the OPEN routine.

PROCEDURE:

1) Open logical file.

2) Load register X with the number of logical file to be used.

3) Call this routine (with JSR).

Possible Errors:

#3 : File not open

#5 : Device Not Present.

#6 : File is not an input file.

EXAMPLE:

```
$1010                                ;Preparation for input
$1010                                ;From logical file 2
$1010      A2 02       LDX #2
$1012      20 C6 FF    JSR CHKIN
```

NAME: CHKOUT

PURPOSE: Open a channel for output

ADDRESS: $FFC9

TRANSFER: X

PREPARATION: (OPEN)

FAULT INDICATION: 0, 3, 5, 7 (see READST)

STACK REQUIREMENT: 4+

REGISTER: A, X

Description:

A logical file number created by the KERNAL routine OPEN can be defined as an output channel. Naturally it has to be an output device otherwise an error will occur and the routine will be aborted.

Before data can be sent to an output device you have to call this routine unless you want to use the screen of the C-16 as an output device. If so and no other output channel are defined, then this routine and the OPEN routine are not needed.

When opening the channel for serial bus this routine automatically sends the LISTEN address specified by the OPEN routine (and possibly also a secondary address).

Procedure:

1) Use the KERNAL OPEN routine to specify a logical file no, a listen address and a secondary address (if required).

2) Load register X with the logical file number used in the OPEN command.

3) Call this routine (via JSR).

Possible Errors:

#3 : Fil not open

#5 : Device Not Present.

#7 : NoT an output file.

EXAMPLE:

```
$1010    A2 03       LDX #3      ;Define logical file 3

$1012    20 C9 FF    JSR CHKOUT  ;as output channel
```

NAME: CHRIN

PURPOSE: Get A Character From Input.

ADDRESS: $FFCF

TRANSFER: A

PREPARATION: (OPEN, CHKIN)

ERRORS: 0 (see READST)

STACK REQUIREMENT: 7+

REGISTER: A, X

Description:

This routine will get a data byte from the channel already set up as the input channel by the KERNAL CHKIN. If CHKIN was not used to define another input channel, a keyboard input is expected. The data byte is returned in the accumulator. After calling, the channel remains open.

Inputs with the keyboard are dealt in a special way. First the cursor is switched on and flashes until a CR (Carriage Return) is typed. All characters in one line (max. 88 characters) are stored in the BASIC input buffer. These characters can be received one at a time by calling this routine for each individual character. When "Carriage Return" is retrieved, the whole line has been processed. When this routine is called again, the procedure is repeated,i.e. flashing of the cursor.

Procedure (from the keyboard):

1) Call this routine (Use JSR).

2) Retrieve a data byte by calling this routine

3) Store Data Byte

4) Check, if it Ii the last data byte (is it a CR?).

5) If not, go to step 2.

EXAMPLE:

```
$1010    A2 00       LDY #$00      ;prepare Y Register
$1012    20 CF FF    JSR CHRIN
$1015    99 DO 0     STA DATA,Y    ;store data
$1018    C8          INY
$1019    C9 13       CMP #CR       ;Is it a "Carriage Return"?
$101B    DO F5       BNE $1012     ;No,fetch next byte
```

Procedure (From Other Devices):

1) Use KERNAL routines OPEN and CHKIN.

2) Call this routine (via JSR).

3) Store data.

EXAMPLE:

```
$1010    20 CF FF     JSR CHRIN
$1013    8D FO 0      STA DATA
```

NAME: CHROUT

PURPOSE: Output A Character

ADDRESS: $FFD2

TRANSFER: A

PREPARATION: (CHKOUT, OPEN)

ERRORS: 0 (see READST)

STACK REQUIREMENT: 8+

REGISTER: A

Description:

This routine will output a character to an already open channel. Use the KERNAL OPEN and CHKOUT routines to set up the output channel before calling this routine. If this call is omitted, data will be sent to default output device (3, screen). The data byte to be sent is loaded in the accumulator and this routine called. The data is then sent to the stipulated output device. After call the channel remains open.

Be careful if this routine is used for data transfer to a serial device, since data will be sent to all open output channels on the bus. If you do'nt want it to happen, all output channels of serial bus ,apart from the required channel, have to be closed with KERNAL routine CLRCHN.

Procedure:

1) If necessary use KERNAL routine CHKOUT (see above).

2) Load data to be sent into accumulator.

3) Call this routine.

EXAMPLE:

```
$1010                                ;Replace BASIC command:CMD 4,"A";
$1010    A2 04       LDX #4          ;Logical File #4
$1012    20 C9 FF    JSR CHKOUT      ;Open output channel
$101     A9 41       LDA #"A"
$1017    20 D2 FF    JSR CHROUT      ;Send character
```

NAME: CIOUT

PURPOSE: Output A Byte Over a Serial Bus

ADDRESS: $FFAB

TRANSFER: A

PREPARATION: LISTEN, (SECOND)

ERRORS: see READST

STACK REQUIREMENT: 5

REGISTER: None

Description:

This routine is used for information transfer to devices on serial bus. By calling this routine a data byte is transferred to serial bus with full serial Handshake. Before calling this routine, LISTEN has to be called to command a device on serial bus to get ready to receive data. (If a secondary address is required, it has to be sent using KERNAL routine SECOND.)

The accumulator is loaded with a Byte which is sent as data via the serial bus. A device has to be ready for data receipt, otherwise the status word will show "Timeout". This routine buffers one character i.e holds the previous character to be sent back. If you call KERNAL routine UNLSN to end data transfer, the buffered character is sent with EOI. Then the command UNLSN is sent to the device.

Procedure:

1) Use KERNAL routine LISTEN (and possibly SECOND).

2) Load one Data Byte in the accumulator.

3) Call this routine to send the data byte.

EXAMPLE:

```
$1010   A9 58      LDA #"X"  ;Send an X to serial bus
$1012   20 A8 FF   JSR CIOUT
```

NAME: CINT

PURPOSE: Initialise Screen Editor And TED chip

ADDRESS: $FF81

TRANSFER: None

PREPARATION: None

ERRORS: None

STACK REQUIREMENT: 4

REGISTER: A, X, Y

Description:

This routine initialises TED Chip in the Commodore 16. Also the KERNAL screen editor is initialised. This routine can be called via a program module of the C-16.

Procedure:

1) Call this routine.

EXAMPLE:

```
$1010   20 81 FF   JSR CINT       ;Initialise TED chip
$1013   20 00 20   JMP RUN        ;Start main program
```

NAME: CLALL

PURPOSE: Close All Files

ADDRESS: $FFE7

TRANSFER: None

PREPARATION: None

ERROR : None

STACK REQUIREMENT: 11

REGISTER: A, X

Description:

This routine closes all open files, the pointers in the open file table are reset and all files closed. The routine CLRCHN will be automatically called for resetting the input/output channels.

Procedure:

1) Call this routine.

EXAMPLE:

```
$1010     20 E7 FF    JSR CLALL   ;Close all data banks and
$1013                             ;Reset I/O channels
$1015     20 00 20    JMP RUN     ;Start main program
```

NAME: CLOSE

PURPOSE: Closing a logical file

ADDRESS: $FFC3

TRANSFER: A

PREPARATION: None

ERRORS: 0, 240 (see READST)

STACK REQUIREMENT: 2+

REGISTER: A, X, Y

Description:

This routine is used for closing a logical file after all inputs/outputs operations on the file are finished. The routine is called after the accumulator is loaded with the logical file number. It is the same number which was used on opening the file with the OPEN routine.

Procedure:

1) Load accumulator with the number of the logical file to be closed

2) Call this routine.

EXAMPLE:

```
$1010                                   ;Close channel 15

$1010    A9 0F        LDA #15

$1012    20 CC FF     JSR CLOSE
```

NAME: CLRCHN

PURPOSE: Clear Input/Output Channels

ADDRESS: $FFCC

TRANSFER: None

PREPARATION: None

ERRORS :

STACK REQUIREMENT: 9

REGISTER: A, X

Description:

This routine is called to clear all open channels and restore the input/output channels to their default values. Normally it is called after opening other input/output channels (e.g. Cassette or Disk) and using them for I/O operations. The default input device no. is 0 (keyboard) and the default output device is 3 (screen).

If one of the channels to be closed is the serial port, an UNTALK signal is sent first to clear the O/P channel or an UNLISTEN is sent to clear the output channel. If this routine is not called (and all connected devices on the serial bus stay active) then several units can receive the same data from the Commodore 16 at the same time.One way to take advantage of this would be for the printer to TALK and for disk to listen. In this way a disk data file can be printed directly.

When using the KERNAL routine CLALL, this routine will be called-up automatically.

Procedure:

1) Call this routine via the JSR-instruction.

EXAMPLE:

```
$1010     20 CC FF     JSR CLRCHN
```

NAME: GETIN

PURPOSE: Get A Character From The Keyboard

ADDRESS: $FFE4

TRANSFER: A

PREPARATION: CHKIN, OPEN

ERRORS: see READST

STACK REQUIREMENT: 7+

REGISTER: A (X,Y)

Description:

The routine takes a character from the keyboard queue and transfers it as an ASCII value to the accumulator. If the queue is empty, the value 0 is loaded in the accumulator. Characters are put in the queue automatically by an interrupt driven keyboard scan routine which calls routine SCNKEY. A maximum of 10 characters can be stored in the keyboard memory. If the memory is full then additional characters are ignored until at least one character has been removed from the queue.

If RS-232 O/P is used then only register A is used and a single character quoted. For checking see READST. If the O/P is the serial bus, the cassette or the screen, call the BASIN-routine.

Procedure:

1) Call this routine with a JSR-instruction.

2) Check, if a 0 is stored in the accumulator (empty memory).

3) Process data.

EXAMPLE:

```
$1010                              ; Wait for a character
$1010    20 E4 FF    JSR GETIN
$1013    C9 00       CMP #0
$1015    F0 F9       BEQ $1010
```

NAME: IOBASE

PURPOSE: Define I/O Memory Page

ADDRESS: $FFF3

TRANSFER: X, Y

PREPARATION: None

ERROR :

STACK REQUIREMENT: 2

REGISTER: X, Y

Description:

This routine sets the X and Y registers to the address of the memory section where the memory mapped I/O devices are located. This address can then be used, together with relative addresses to access the memory mapped I/O devices of the Commodore C-16. Register X contains the lower order address byte and register Y the upper order address byte.

With this routine the compatibility between the VC-20, the C-64 and the C-16 is guaranteed. If the input/output registers for a machine code program are set by calling this routine, they will also be compatible with future versions of the Commodore C-16 with regard to KERNAL and BASIC.

Procedure:

1) Call this routine with the JSR-instruction.

2) Store registers X and Y in consecutive positions.

3) Load the Y register with the offset

4) Access the input/output location.

EXAMPLE: (only PLUS/4)

```
$1010                ;Data direction register of User-Port on 0
$1010   20 F3 FF     JSR IOBASE
$1013   86 D0        STX POINT     ;Place BASIS-register
$1015   84 D1        STY POINT+1
$1017   A2 02        LDY #2
$1019   A9 00        LDA #0        ;Offset for DDR of User-Port
$101B   91 D0        STA (POINT),Y;Put DDR on 0
```

NAME: IOINIT

PURPOSE: Initialise I/O Devices

ADDRESS: $FF84

TRANSFER: None

PREPARATION: None

ERROR :

STACK REQUIREMENT:None

REGISTER: A,X,Y

Description:

With this routine all input/output devices and routines are initialised. It is normally called as part of the initialising program module of the Commodore 16.

Procedure:

1) Call this routine with JSR

EXAMPLE:

```
$1010    20 84 FF     JSR IOINIT
```

NAME: LISTEN

PURPOSE: Command A Device To Listen

ADDRESS: $FFB1

TRANSFER: A

PREPARATION: None

ERROR : see READST

STACK REQUIREMENT: None

REGISTER: A

Description:

This routine commands a device on serial bus to receive data. A device number between 0 and 31 is loaded into the accumulator before calling this routine. LISTEN will OR the number bit by bit to convert to a LISTEN address. The addressed device will then go into listen mode and be ready to accept data.

Procedure:

1) Load the accumulator with the number of device commanded to listen.

2) Call this routine with JSR.

EXAMPLE:

```
$1010                       ;Unit no. 8 on LISTEN

$1010    A9 08        LDA #8

$1012    20 B1 FF     JSR LISTEN
```

NAME: LOAD

PURPOSE: Load RAM From Device

ADDRESS: $FFD5

TRANSFER: A,X,Y

PREPARATION: SETLFS, SETNAM

ERRORS: 0,4,5,8,5, READST

STACK REQUIREMENT: None

REGISTER: A,X,Y

Description:

This routine directly loads data byte from an input device into the memory of the Commodore 16. It can also be used for comparing data from a device with the data in the memory while the data stored in RAM remains unchanged (VERIFY).

Accumulator is set to 0 for loading or to 1 for VERIFY. If input device was OPENed with a secondary address of 3, the header will be ignored. In that case registers X and Y must contain the start address for loading. If secondary address 1, 0 or 2 are selected, the data will be loaded into the memory from the position specified by the header. This routine returns the address of the highest RAM location which was loaded.

Before calling this routine KERNAL routines SETLFS and SETNAM have to be called-up.

A LOAD with keyboard (0), RS-232 (2) or screen (3) is not possible.

Procedure:

1) Call SETLFS and SETNAM. If a relocated load is required, use routine SETLFS to send the secondary address 0.

2) Load acc. with 0 for LOAD or 1 for VERIFY.

3) If loading is to relocated address, registers X and Y must be set to the start address for the load.

4) Call this routine using the JSR-instruction.

EXAMPLE:

```
$1010                                   ;Load a program from cassette
$1010   A9 01     LDA #DEVICE1          ;Set device no.
$1012   A2 00     LDX "FILENO           ;Set logical file no.
$1014   A0 00     LSY CMD1              ;Set secondary address
$1016   20 BA FF  JSR SETLFS
$1019   A9 0C     LDA #NAME1-NAME       ;Load acc. with no. of
$1019                                   ; characters in file name
$101B   A2 32     LDX #<NAME            ;Load X and Y with address
$101D   A0 10     LDY #>NAME            ;of program name ($1032)
$101F   20 BD FF  JSR SETNAM
$1022   A9 00     LDA #0                ;Set Flag for load
```

```
$1024   A2 FF     LDX #$FF          ;Default start

$1026   A0 FF     LDY #$FF

$1028   20 D5 FF  JSR LOAD

$102B   86 2D     STX VARTAB        ;End address of program

$102D   84 2E     STY VARTAB+1

$102F   4C 00 20  JMP START

$1032   50 52 4F  .BYT "FILE NAME"

$1035   47 52 41

$1038   4D 4D 4E

$103B   41 4D 45
```

NAME: MEMBOT

PURPOSE: Set Bottom Of Memory

ADDRESS: $FF9C

TRANSFER: X, Y

PREPARATION: None

ERROR : None

STACK REQUIREMENT: None

REGISTER: X, Y

Description:

This routine is used to set the bottom memory pointer. If, on call up of this routine, the accumulator carry bit is set then the pointer to the bottom RAM byte is returned in registers X and Y. The pointer start value of the Commodore C-16 without additional memory expansion is $1000 (4096 decimal). If the accumulator carry bit is 0 on call up of this routine, then the values of registers X and Y are transferred to the low and high bytes of the pointer to the start of RAM.

Procedure: (Read the bottom of RAM):

1) Set carry flag

2) Call this routine

Procedure (Set the bottom of memory):

1) Clear the carry flag

2) Call this routine

EXAMPLE

```
$1010                                ;Move bottom of memory
                                     ; up 1 page (256 Bytes)

$1010    38          SEC             ;Read memory bottom

$1011    20 9C FF    JSR MEMBOT

$1014    E8          INY

$1015    18          CLC             ;Set memory bottom to new value

$1016    20 9C FF    JSR MEMBOT
```

NAME: MEMTOP

PURPOSE: Setting pointer to the top of memory

ADDRESS: $FF99

TRANSFER: X, Y

PREPARATION: None

ERRORS: None

STACK REQUIREMENT: 2

REGISTER: X, Y

Description:

This routine is used to set the top of RAM. If, on call up of this routine,the carry bit is set, then the pointer to the top of RAM is loaded into registers X and Y. If the accumulator carry bit is clear on call up of this routine then the contents of the registers X and Y are loaded in the top memory pointer, changing the top of memory.

EXAMPLE:

```
$1010                                ;Release RS-232 Buffer
$1010   38          SEC
$1011   20 99 FF    JSR MEMTOP      ;Read memory top
$1014   88          DEX
$1015   18          CLC
$1016   20 99 FF    JSR MEMTOP      ;Set new top of memory
```

NAME: OPEN

PURPOSE: Opening A Logical File

ADDRESS: $FFC0

TRANSFER: None

PREPARATION: SETLFS, SETNAM

ERRORS: 1,2,4,5,6,240,READST

STACK REQUIREMENT: None

REGISTER: A, X, Y

Description:

This routine is used to open a logical file. After opening a logical file, it can be used for I/O operations. This routine is called by most KERNAL input/output routines to create the relevant logical data files. No parameters are required setting, but KERNAL routines SETLFS and SETNAM have to be called beforehand.

Procedure:

1) Use routine SETLFS.

2) Use routine SETNAM.

3) Call this routine.

EXAMPLE:

```
$1010                                ;This routine simulates the BASIC
$1010                                ;command:OPEN 15,8,15,"IO"
$1010   A9 02    LDA #NAME1-NAME     ;Length of file name (2)
$1012   A2 25    LDX #<NAME          ;Address of file name in
$1014   A0 10    LDY #>NAME          ;X and Y Reg. ($1025)
$1016   20 BD FF JSR SETNAM
$1019   A9 0F    LDA #15
$101B   A2 08    LDX #8
$101D   A0 0F    LDY #15
$101F   20 BA FF JSR SETLFS
$1022   20 C0 FF JSR OPEN
$1025   49 30    .BYT "IO"
```

NAME: PLOT

PURPOSE: Read/Set Cursor Position

ADDRESS: $FFF0

TRANSFER: A,X,Y

PREPARATION: None

ERRORS: None

STACK REQUIREMENT: 2

REGISTER: A,X,Y

Description:

If accumulator carry flag is set when this routine is called, then the present cursor position on the screen (in X and Y coordinates) is returned in registers X and Y. X is the row number (0-24) of the cursor position, Y the column number (0-39). If on call, the carry bit is clear, the cursor moves to the position X, Y (according to registers X and Y).

Procedure (reading cursor position):

1) Set carry flag

2) Call this routine

3) Read X and Y positions from registers X and Y resp.

Procedure (placing cursor position):

1) Clear carry flag

2) Write required cursor position in registers X and Y.

3) Call this routine.

EXAMPLE:

```
$1010                              ;Place cursor in
$1010                              ;row 10, column 5
$1010    A2 0A     LDX #10
$1012    A0 05     LDY #5
$1014    18        CLC
$1015    20 F0 FF  JSR PLOT
```

NAME: RAMTAS

PURPOSE: RAM Test

ADDRESS: $FF87

TRANSFER: A,X,Y

PREPARATION: None

ERRORS: None

STACK REQUIREMENT: 2

REGISTER: A,X,Y

Description:

With this routine, RAM is tested and the top/bottom memory pointers set. Also the memory locations $0000 - $0101 and $0200 - $07FF are cleared. Furthermore the cassette buffer is initialised and the screen start positioned at $0C00. Normally this routine is called as part of the power on module of the Commodore 16.

Procedure:

1) Call this routine.

EXAMPLE:

```
$1010    20 87 FF    JSR RAMTAS
```

NAME: RDTIM

PURPOSE: Read system clock

ADDRESS: $FFDE

TRANSFER: A,X,Y

PREPARATION: None

ERROR : None

STACK REQUIREMENT: 2

REGISTER: A,X,Y

Description:

This routine is used to read the system clock. The clock resolution is 1/60 s. This routine returns 3 bytes. The accumulator contains the most significant byte, the X-register the next significant byte and the Y-register the least significant Byte.

Procedure:

1) Call this routine.

EXAMPLE:

```
$1010   20 DE FF    JSR RDTIM

$1013   84 D0       STY TIME

$1015   86 D1       STX TIME+1

$1017   85 D2       STA TIME+2
```

NAME: READST

PURPOSE: Read Status Word

ADDRESS: $FFB7

TRANSFER: A

PREPARATION: None

ERRORS: None

STACK REQUIREMENT: 2

REGISTER: A

Description:

This routine returns the current status of the I/O devices in the accumulator. Normally this routine is called after new communication to an I/O device. It gives you information about device status or errors which occurred during input/output.

The bits transferred to the accumulator contain the following information:

ST BIT POSITION	ST NUMB. VALUE	READ FROM CASSETTE	SERIAL READ/ WRITE	TAPE LOAD/VERIFY
0	1		Write Timeout	
1	2		Read Timeout	
2	4	Short Block		Short Block
3	8	Long Block		Long Block
4	16	Unrecoverable Read Error		Unrecoverable Read Error
5	32	Checksum Error		Checksum Error
6	64	End Of file	End of line	
7	128	Tape end	Device not present	End Of Tape

Procedure:

1) Call this routine.

2) Decode program information in registre A.

EXAMPLE:

```
$1010                              ;Check for file-end
$1010                              ;during reading
$1010   20 B7 FF     JSR READST
$1013   49 40        AND #64       ;Test for EOF (End Of File)
$1015   D0 25        BNE EOF       ;Branch when placed
```

NAME: RESTOR

PURPOSE: Reset normal status of system

ADDRESS: $FF8A

PREPARATION: None

ERROR : None

STACK REQUIREMENT: 2

REGISTER: A,X,Y

Description:

This routine restores the default values of all system vectors used in KERNAL and BASIC routines and Interrupts are reset. The individual system vectors can be read and prepared with KERNAL-routine VECTOR.

Procedure:

1) Call this routine.

EXAMPLE:

```
$1010    20 8A FF    JSR RESTOR
```

NAME: SAVE

PURPOSE: Transfer RAM Contents To A DEVICE

ADDRESS: $FFD8

TRANSFER: A,X,Y

PREPARATION: SETLFS, SETNAM

ERRORS: 5,8,9,READST

STACK REQUIREMENT: None

REGISTER: A,X,Y

Description:

This routine saves a section of memory. The memory is saved from an indirect address in the zero page given in the accumulator to the fixed address in registers X and Y. Before calling this routine, the routines SETLFS and SETNAM must be called. Saving on device 1 (cassette), file name is not necessary (Though recommended). If you try to save on a different device without file name, an error will result.

It is not possible to save on device 0 (keyboard), 2 (RS-232) and 3 (screen), as an error will occur and the routine aborted.

Procedure:

1) User routines SETLFS and SETNAM (if not to be saved on tape without file name.)

2) Load two consecutive locations of zero page with the pointer to the start of save (as a standard in the 7501 the lower Byte comes first and then the higher Byte).

3) Load address of pointer in zero page in the accumulator.

4) Load low and high byte respetively of end address of save in registers X and Y.

5) Call this routine.

EXAMPLE:

```
$1010 A9 01                        ;Device 1 (cassette)
$1012    20 BA FF    JSR SETLFS
$1015    A9 00       LDA #0        ;No name
$1017    20 BD FF    JSR SETNAM
$101A    A9 00       LDA START     ;Load start address ($3C00)
$101C    85 2B       STA TXTTAB    ;(LOW BYTE)
$101E    A9 3C       LDA START+1
$1020    85 2C       STA TXTTAB+1  ;(HIGH BYTE)
$1022    A2 00       LDX END+1     ;Load end address ($4000) (LOW)
$1024    A0 40       LDY END+1     ;(HIGH BYTE)
$1026    A9 2B       LDA TXTTAB    ;Load acc. with zero page addr.
$1029    20 D8 FF    JSR SAVE
```

NAME: SCNKEY

PURPOSE: Keyboard Scan

ADDRESS: $FF9F

TRANSFER: None

PREPARATION: IOINIT

ERRORs: None

STACK REQUIREMENT: 5

REGISTER: A,X,Y

Description:

This routine will scan the Commodore 16's keyboard and check for pressed key. If yes, which keys were pressed. This is the same routine which is called at each Interrupt. After pressing a key, the relevant ASCII-value is stored in the keyboard memory.

This routine is only called if the normal IRQ-interrupt is ignored.

Procedure:

1) Call this routine.

EXAMPLE:

```
$1010    20 9F FF    JSR SCNKEY      ;Keyboard scan

$1013    20 E4 FF    JSR GETIN       ;Fetch character

$1016    C9 00       CMP #0          ;Key pressed?

$1018    F0 F6       BEQ GET         ;Yes, scan again

$101A    20 D2 FF    JSR CHROUT      ;Print character
```

NAME: SCREEN

PURPOSE: Identify Screen Format

ADDRESS: $FFED

TRANSFER: X,Y

PREPARATION: None

STACK REQUIREMENT: 2

REGISTER: X,Y

Description:

This routine shows the screen format,e.g. 40 columns and 25 lines in Y. It can be used to determine on which machine a program is running. This function was introduced for the Commodore 64 to make your programs more easily compatible with other computers.

Procedure:

1) Call this routine.

EXAMPLE:

```
$1010     20 ED FF     JSR SCREEN

$1013     86 D0        STX MAXCOL

$1015     84 D1        STY MAXROW
```

NAME: SECOND

PURPOSE: Transfer of secondary address for LISTEN

ADDRESS: $FF93

TRANSFER: A

PREPARATION: LISTEN

ERRORS: see READST

STACK REQUIREMENT: 8

REGISTER: A

Description:

This routine sends a secondary address to an I/O device after calling the LISTEN routine and device commanded to LISTEN. This routine cannot be used for transferring a secondary address after calling the TALK routine.

Preparation:

1) Load secondary address to be sent in the accumulator, OR it with $60

2) Call this routine.

EXAMPLE:

```
$1010                                   ;Address unit no. 8
$1010                                   ;(Floppy) with secondary
$1010                                   ;address 15
$1010   A9 08       LDA #8
$1012   49 60       EOR #96
$1014   20 B1 FF    JSR LISTEN
$1017   A9 0F       LDA #15
$1019   20 93 FF    JSR SECOND
```

NAME: SETLFS

PURPOSE: Opening A Logical Data File

ADDRESS: $FFBA

TRANSFER: A,X,Y

PREPARATION: None

ERRORS: None

STACK REQUIREMENT: 2

REGISTER: None

Description:

This routine will set the logical file no., device address and a secondary address (command number) for other KERNAL routines.
The logical file no. is used by the system as a kind of key for the file table, created by the OPEN routine. Device addresses can range from 0-31. The following codes are used .

ADDRESS:	ITEM:
0	Keyboard
1	Cass. Recorder
2	RS-232C
3	Screen
4	Printer on Serial Bus
8	Disc Drive

A device number of 4 or above refers automatically to devices on the serial bus.

A command to the device is sent as secondary address via the serial bus after the device number has been sent during the serial handshake. If no secondary address is being sent, the Y register must be set to 255.

Procedure:

1) Load logical file number in accumulator.

2) Load device number into X register.

3) Load command into Y register.

4) Call this routine.

EXAMPLE:

```
$1010                                   ;Logical file 32, device no 4

$1010                                   ;and no command no.

$1010     A9 20       LDA #32

$1012     A2 04       LDX #4

$1014     A0 FF       LDY #255

$1016     20 BA FF    JSR SETLFS
```

NAME: SETMSG

PURPOSE: Print Out Of System Messages

ADDRESS: $FF90

TRANSFER: A

PREPARATION: None

ERRORS: None

STACK REQUIREMENT: 2

REGISTER: A

Description:

This routine controls the printing of error and control messages by the KERNAL. When called, the errors message or control message can be selectedd according to the contents of the accumulator.
Error message example -FILE NOT FOUND.
Control message example - PRESS PLAY ON TAPE

Bit 7 = 1---Error Message

Bit 6 = 1---Control Message

Procedure:

1) Set accumulator to required value.

2) Call this routine.

EXAMPLE:

```
$1010    A9 40      LDA #$40          ;Control messages
$1012    20 90 FF   JSR SETMSG
$1015    A9 80      LDA #$80          ;Error messages
$1017    20 90 FF   JSR SETMSG
$101A    A9 00      LDA #0            ;All KERNAL messages off
$101C    20 90 FF   JSR SETMSG
```

NAME: SETNAM

PURPOSE: Set Up File Name

ADDRESS: $FFBD

TRANSFER: A,X,Y

PREPARATION: None

ERRORS: None

REGISTER: None

Description:

This routine sets up the file names for the routines OPEN, SAVE or LOAD. The no. of characters in the file name is loaded into the accumulator. The address of the file name is loaded into registers X and Y. The standard lower byte/higher byte format applies. The address can be any valid system memory address from which the file name is stored as a string. If no file name is required, the accumulator is loaded with 0. In this case registers X and Y can be set to any memory address.

Procedure:

1) Load length of file name into accumulator.

2) Load lower value address byte of file name into X register.

3) Load higher value address byte of file name into Y register.

4) Call this routine.

EXAMPLE:

```
$1010    A9 04      LDA #NAME1-NAME  ;Load length of file name
$1012    A2 1A      LDX #<NAME       ;Load address of file name
$1014    A0 10      LDY #>NAME
$1016    20 BD FF   JSR SETNAM
```

NAME: SETTIM

PURPOSE: Set System Clock

ADDRESS: $FFDB

TRANSFER: A,X,Y

PREPARATION: None

ERRORS: None

STACK REQUIREMENT: 2

REGISTER: None

Description:

A system clock maintained by the interrupt routine is updated every 1/60 s ("Jiffy"). The clock is 3 Bytes "long", so that it can show up to 5,184,000 "Jiffies" (24 hrs.). Then the clock resets to 0. Before calling this routine, the accumulator has to be loaded with the most significant (highest) byte, the X Register with the next significant byte and the Y Register with the least significant byte of the start up time (in Jiffies).

Procedure:

1) Load the most significant Byte of the 3 Byte number in the accumulator.

2) Load next Byte in the X Register.

3) Load the least significant Byte in the Y Register.

4) Call this routine.

EXAMPLE:

```
$1010                       ;Set clock to 10 minutes
$1010                       ;(=3600 Jiffies)
$1010   A9 00      LDA #0           ;Most significant Byte
$1012   A2 0E      LDX #>3600       ;Next Byte
$1014   A0 10      LDY #<3600       ;Lowest Byte
$1016   20 DB FF   JSR SETTIM
```

NAME: SETTMO

PURPOSE: Set Timeout Flag for Serial bus

ADDRESS: $FFA2

TRANSFER: A

PREPARATION: None

ERRORS: None

STACK REQUIREMENT: 2

REGISTER: None

Description:

This routine is exclusively used with an additional Bus-Card and is practically of no importance for the Commodore 16.

With this routine the Timeout-Flag for the IEC-Bus is set. When this is done the Commodore C-16 waits 64 ms for the message from a device on IEC-Bus. If the device does not answer to the DAV-Signal (valid data address) of the Commodore 16 within this period of time then the computer recognizes an error and leaves the Handshake sequence. If on call up of this routine Bit 7 in the accumulator is set to 0, then Timeouts are effective. Accordingly Timeouts are ineffective if Bit 7 is set to 1.

Procedure (Set Timeout-Flag):

1) Set Bit 7 of accumulator to 0.

2) Call this routine.

Procedure (Clearing Timeout Flag):

1) Set Bit 7 of accumulator to 1.

2) Call this routine.

EXAMPLE:

```
$1010                                   ;Put Timeout-Flag

$1010    A9 00       LDA #0

$1012    20 A2 FF    JSR SETTMO
```

NAME: STOP

PURPOSE: STOP Key Scan

ADDRESS: $FFE1

TRANSFER: A

PREPARATION: NONE

ERRORS: None

STACK REQUIREMENT: None

REGISTER: A,X

Description:

If during an UDTIM call up the STOP key was pressed, then after call-up of this routine the Zero Flag is set. Furthermore the channels are reset to the default values. All other Flags remain unchanged. If the STOP key was not pressed, the accumulator contains 1 byte of the last keyboard scan, so the operator can also check if certain other keys were pressed.

Procedure:

1) Before this routine, UDTIM has to be called.

2) Call this routine.

3) Check for Zero Flag.

EXAMPLE:

```
$1010    20 EA FF    JSR UDTIM       ;Check for STOP
$1013    20 E1 FF    JSR STOP
$1016    F0 FB       BNE LOOP        ;Wait for STOP key
```

NAME: TALK

PURPOSE: Command A Device On Serial Bus To Talk

ADDRESS: $FFB4

TRANSFER: A

PREPARATION: None

ERRORS: see READST

STACK REQUIREMENT: 8

REGISTER: A

Description:

In order to use this routine, a device no. between 0 and 31 has to be loaded into the accumulator. When called,this routine does a logical OR bit by bit to convert the device no. to talk address. This data is then sent as command via the serial bus.

Procedure:

1) Load accumulator with device no

2) Call this routine.

EXAMPLE:

```
$1010                          ;TALK command for device 4

$1010    A9 04      LDA #4

$1012    20 B4 FF   JSR TALK
```

NAME: TKSA

PURPOSE: Send a secondary address to a device commanded to talk

ADDRESS: $FF96

TRANSFER: A

PREPARATION: TALK

ERRORS: see READST

STACK REQUIREMENT: 8

REGISTER: A

Description:

This routine sends a secondary address via the serial bus to a TALK device. This routine must be called with a number between 4 and 31 in the accumulator. This no. is then sent as secondary address command via the serial bus. It is vital to call the TALK routine before hand. TKSA is not effective after LISTEN.

Procedure:

1) Use TALK-routine.

2) Load secondary address into accumulator, OR with 96.

3)Call this routine.

EXAMPLE:

```
$1010                    ;Tell device no # 4 to talk with command 7

$1010    A9 04      LDA #4

$1012    20 B4 FF   JSR TALK

$1015    A9 07      LDA #7

$1017    49 60      EOR #96

$1019    20 96 FF   JSR TKSA
```

NAME: UNTLK

PURPOSE: Send an UNTALK command

ADDRESS: $FFAB

TRANSFER: None

PREPARATION: None

ERRORS: see READST

STACK REQUIREMENT: 8

REGISTER: A

Description:

With this routine an UNTALK command is sent via serial bus. All commands which received a TALK command before stop data transfer.

Procedure:

1) Call this routine.

EXAMPLE:

```
$1010   20 AB FF    JSR UNTLK
```

NAME: VECTOR

PURPOSE: Manage RAM Vectors

ADDRESS: $FF8D

TRANSFER: X,Y

PREPARATION: None

ERROR MESSAGE: None

STACK REQUIREMENT: 2

REGISTER: A,X,Y

Description:

This routine manages all jump vectors stored in RAM. If on call up of this routine, the accumulator carry bit is set, then the current contents of the RAM Vectors are stored in a table whose address is given by the contents of registers X and Y.

When working with this routine you have to be extremely careful. First of all the entire vector contents should be read in the user area, the required vectors changed and then copy the content back to the system vectors.

Procedure (Reading the System RAM Vectors):

1) Set Carry

2) Load registers X and Y with the required Vector Address.

3) Call this routine.

Procedure (Load System RAM Vectors):

1) Clear Carry

2) Set registers X and Y to the RAM Address Vector table to be loaded.

3) Call this routine.

EXAMPLE:

```
$1010                              ;Change input routine
$1010    A2 30       LDX #<USER
$1012    A0 10       LDY #>USER
$1014    38          SEC
$1015    20 8D FF    JSR VECTOR    ;Read old vectors
$1018    A9 00       LDA #<MYINP   ;Own input routine
$101A    8D 3A 10    STA USER+10
$101D    A9 20       LDA #>MYINP
```

```
$101F    8D 3B 10    STA USER+11

$1022    A2 30       LDX #<USER

$1024    A0 10       LDY #>USER

$1026    18          CLC

$1027    20 8D FF    JSR VECTOR    ;Change vectors
```

Listed below, you will find a list of error messages which can appear when working with the KERNAL routines. If one of these errors occur, the carry bit of the accumulator is set and the no. of the error message transferred to the accumulator.

Some KERNAL I/O routines do not work with these codes of error messages. Instead the errors are identified by the KERNAL routine READST.

NUMBER:	MEANING:
0	Routine finished by STOP key
1	Too many open files
2	File already open
3	File not open
4	File not found
5	Device not present
6	File is not an input file
7	File is not an output file
8	File name is missing
9	Illegal device no

4.5 MEMORY MAP

ADDRESS (hex)	ADDRESS (dec)	LABEL	DESCRIPTION
$0000	0	PDIR	7501 Data Direction Register
$0001	1	PORT	7501 8 Bit I/O-Port
$0002	2	SRCHTK	Flag for loops
$0003-0004	3-4	ZPVEC1	New Start Address (RENUMBER)
$0005-0006	5-6	ZPVEC2	Step Width (RENUMBER)
$0007	7	CHARAC	Search Character
$0008	8	ENDCHR	Flag:Searching for inverted comma at end of a String
$0009	9	TRMPOS	Screen column from last TAB
$000A	10	VERCK	Flag: 0=LOAD, 1=VERIFY
$000B	11	COUNT	Input buffer counter, No. of elements
$000C	12	DIMFLG	Flag:Standard-Field Dimensioning
$000D	13	VALTYP	Variable Flag: $FF=String $00=Numerical
$000E	14	INTFLG	Numeric Flag: $80=Integer $00=Floating Point

$000F	15	DORES	Flag:Data Scan
$0010	16	SUBFLG	Flag:User Function Call-Up
$0011	17	INPFLG	Flag: $00=INPUT, $40=GET $98= READ
$0012	18	TANSGN	Flag: ATN Sign/Comparison
$0013	19	CHANNL	Flag:Current I/O Prompt
$0014-0015	20-21	LINNUM	Store For BASIC Integers
$0016	22	TEMPPT	Pointer: Temporary String Stack
$0017-0018	23-24	LASTPT	Last Temporary String Vector
$0019-0021	25-33	TEMPST	Stack For Temp. String Vector
$0022-0023	34-35	INDEX1	Store For Auxiliary Pointer 1
$0024-0025	36-37	INDEX2	Store For Auxiliary pointer 2
$0026	38	RESHO	Product Area For Multiplication
$0027	39	RESMOH	" " " "
$0028	40	RESMO	" " " "
$0029	41	RESLO	" " " "
$002A	42		
$002B-002C	43-44	TXTTAB	Pointer: Start BASIC
$002D-002E	45-46	VARTAB	Pointer: Start BASIC Variables
$002F-0030	47-48	ARYTAB	Pointer: Start of BASIC Arrays
$0031-0032	49-50	STREND	Pointer: End of Arrays

$0033-0034	51-52	FRETOP	Pointer: Start Strings (Moving Down)
$0035-0036	53-54	FRESPC	Auxiliary Pointer for Strings
$0037-0038	55-56	MEMSIZ	Pointer: Top Of RAM Available For Basic
$0039-003A	57-58	CURLIN	Current BASIC Line No.
$003B-003C	59-60	TXTPTR	Previous BASIC Line No.
$003D-003E	61-62	FNDPNT	Pointer:BASIC Statement For CONT
$003F-0040	63-64	DATLIN	Current DATA Line No.
$0041-0042	65-66	DATPTR	Pointer: Current DATA Address
$0043-0044	67-68	INPPTR	Vector: INPUT Routine
$0045-0046	69-70	VARNAM	Current BASIC Variable Name
$0047-0048	71-72	VARPNT	Current Variable Address
$0049-004A	73-74	FORPNT	Variable Pointer For FOR/NEXT
$004B-004C	75-76	OPPIR	Y Save;Operator Save Basic Pointer Save
$004D	77	OPMASK	Comparison Mask: 1= Larger, 2= Equal, 4= Smaller
$004E-004F	78-79	DEFPNT	Pointer:Variable Of DEF FN
$0050-0051	80-81	DSCPNT	Pointer:String Descriptor
$0052	82		
$0053	83	HELPER	Flag: HELP or LIST
$0054-0056	84-86	JMPER	Jump Vector For Functions

$0057-0060	87-96	TEMPF1	Numeric Work Area
$0061	97	FACEXP	Accumulator 1-Exponent
$0062-0065	98-101	FACHO	" 1-Mantissa
$0066	102	FACSGN	" 1-Sign
$0067	103	SGNFLG	Pointer:Series Evaluation Const.
$0068	104	BITS	Accumulator 1 Hi-Order(Overflow)
$0069	105	ARGEXP	" 2 Exponent
$006A-006D	106-109	ARGHO	" 2 Mantissa
$006E	110	ARGSGN	" Sign
$006F	111	ARISGN	Sign Comp. Acc 1 Vs Acc 2
$0070	112	FACOV	Accumulator 1: Lo Order Rounding
$0071-0072	113-114	FBUFPT	Pointer: Cassette Buffer
$0073-0074	115-116	AUTINC	Increment On AUTO-Command, 0=Off
$0075	117	MVDFLG	Flag:1=10K Reserved For Graphics
$0076-0078	118-120	KEYNUM	Temporary store for MID$-Command
$0079-007B	121-123	DSDESC	Descriptor for DS$
$007C-007D	124-125	TOS	Top Of Run-Time Stacks
$007E-007F	126-127	TMPTON	Working Area (Sound)
$0080	128	VOICNO	" " "
$0081	129	RUNMOD	Flag For RUN ($00=No, $80=Yes)
$0082	130	POINT	Flag for DOS-Command

$0083	131	GRAPHM	Current Graphics Mode ($00=Text, $20=Hires, $60=Split Hires, $A0= Multicolour,$E0=Split Multi Col.
$0084	132	COLSEL	Current Chosen Colour
$0085	133	MC1	Multicolour-Colour 1
$0086	134	FG	Foreground-Colour
$0087	135	SCXMAX	Max. No. Of Columns
$0088	136	SCYMAX	Max. No. Of Rows
$0089	137	LTFLAG	Flag: PAINT Left
$008A	138	RTFLAG	Flag: Paint Right
$008B	139	STOPNB	Flag: Paint Stop ($00=same,$80= Different) As Background Colour
$008C-008D	140-141	GRAPNT	Pointer: Bit Map Colour
$008E	142	VTEMP1	Temporary Intermediate Memory
$008F	143	VTEMP2	" " "
$0090	144	STATUS	KERNAL I/O Status Word:ST
$0091	145	STKEY	Flag: STOP/RVS Key
$0092	146	SPVERR	Temporary Memory
$0093	147	VERFCK	Flag: 0=LOAD, 1=VERIFY
$0094	148	C3PO	Flag: Serial Bus Character In Buffer ($00=No, $80=Yes)
$0095	149	BSOUR	Ch. In Buffer For serial Bus
$0096	150	RSAV	Temporary memory for BASIC

$0097	151	LDTND	No. Of Open Files / Index For File Tables
$0098	152	DFLTN	Default Input Device(Norm. 0)
$0099	153	DFLTO	Default O/P Device (Norm. 3)
$009A	154	MSGFLG	Flag: $80=BASIC-Direct Mode, $C0=M/C Monitor, $00=Program
$009B-009C	155-156	SAL	Pointer:Tape Buff./Scroll Screen
$009D-009E	157-158	EAL	Pointer:Tape End/Program End
$009F-00A0	159-160	T1	Temporary Memory
$00A1-00A2	161-162	T2	" "
$00A3-00A5	163-165	TIME	Clock (approx. 1/60 s)
$00A6	166	R2D2	Register for serial bus
$00A7	167	TPBYTE	Register for cassette routine
$00A8	168	BSOUR1	Register for serial bus
$00A9	169	FPVERR	Temporary Colour-Vector
$00AA	170	DCOUNT	Register for cassette routine
$00AB	171	FNLEN	Length of file name
$00AC	172	LA	Logical file number
$00AD	173	SA	Current secondary address
$00AE	174	FA	Current device no.
$00AF-00B0	175-176	FNADR	Current file name
$00B1	177	ERRSUM	Cass. error counter

$00B2-00B3	178-179	STAL	I/O Start Address
$00B4-00B5	180-181	MEMUSS	Basic loading address
$00B6-00B7	182-183	TAPEBS	Pointer:Load end addr. (tape)
$00B8-00B9	184-185	TMP2	Address for VECTOR
$00BA-00BB	186-187	WRBASE	Pointer: Cassette buffer data.
$00BC-00BD	188-189	IMPARM	Pointer : String for Primms
$00BE-00BF	190-191	FETPTR	Pointer:Long Fetch Routine
$00C0-00C1	192-193	SEDSAL	Temp. Mem.:Scroll Register
$00C2	194	RVS	Flag: RVS-Ch.($12=Yes,$00=No)
$00C3	195	INDX	Pointer: End of line for input
$00C4-00C	196-197	LSXP	Cursor X/Y-Position for input
$00C6	198	SFDX	Flag: Pressed key ($40=none)
$00C7	199	CRSW	Flag: INPUT or GET from K/B
$00C8-00C9	200-201	PNT	Pointer:Current scr. line add.
$00CA	202	PNTR	Cursor position current line
$00CB	203	QTSW	Flag: Editor in inverted comma mode (0=No)
$00CC	204	SEDT1	Length of current screen line
$00CD	205	TBLX	Current pos.of cursor line no
$00CE	206	DATAX	Last character (I/O)
$00CF	207	INSRT	Flag:INST-Mode.>0=No of ins.

$00D0-00D7	208-215		Res. for speech synthesizer
$00D8-00E8	218-232		Res. for application software
$00E9	233	CIRSEG	Working area
$00EA-00EB	234-235	USER	Pointer:Current screen colour
$00EC-00EE	236-238	KEYTAB	Vector: K.B. Decoding Tab.$E026
$00EF	239	NDX	No of char.in key board buffer
$00F0	240	STPFLG	Flag:Pause (CTRL-S/T)
$00F1-00F2	241-242	TO	Register for M/C. Monitor
$00F3	243	CHRPTR	Zero-Page-Address for M/C.Mon.
$00F4	244	BUFEND	" " " " " "
$00F5	245	CHKSUM	Register for check sum
$00F6	246	LENGTH	
$00F7	247	PASS	
$00F8	248	TYPE	Block-Type
$00F9	249	USEKDY	bit7=1: Write, bit6=1: read
$00FA	250	XSTOP	Register for X on STOP Key Test
$00FB	251	CURBNK	Present Bank-Configuration
$00FC	252	XON	Character to be sent for X-On
$00FD	253	XOFF	" " " " X-Off
$00FE	254	SEDT2	Work area for Editor
$00FF	255	LOFBUF	

$0100-010F	256-271	FBUFFER	
$0110	272	SAVEA	Intermed.mem.for SAVE & RESTORE
$0111	273	SAVEY	" " " " " "
$0112	274		" " " " " "
$0113-0122	275-289	COLKEY	Col./Luminiscence Table in RAM
$0123	290		
$0124-01FF	291-511	SYSSTK	Processor Stack
$0200-0258	512-600	BUF	Basic I/P or Monitor Buffer
$0259-025A	601-602	OLDLIN	Previous BASIC Line No.
$025B-025C	603-604	OLDTXT	Pointer: BASIC Command for CONT
$025D-02AC	605-684		BASIC/DOS Work Area
$025D	605	XCNT	DOS Loop Counter
$025E-026D	606-621	FNBUFR	Area for file names
$026E	622	DOSF1L	Length of first file name
$026F	623	DOSDS1	DOS (DEVICE 1)
$0270-0271	624-625	DOSF1A	Address of first file name
$0272	626	DOSF2L	Length of second file name

$0273	627	DOSFA	DOS (Device 2)
$0274-0275	628-629	DOSF2A	Address of second file name
$0276	630	DOSLA	DOS logical address
$0277	631	DOSFA	" device address
$0278	632	DOSSA	" secondary address
$0279-027A	633-634	DODDID	" Disk-ID
$027B	635	DIDCHK	ID-Flag
$027C	636	DOSSTR	DOS Output Buffer
$027D-02AC	637-684	DOSSPC	" Work Area
$02AD-02AE	685-686	XPOS	Current X Pos:Graphic Cursor
$02AF-02B0	687-688	YPO	Current Y Pos:Graphics Cursor
$02B1-02B2	689-690	XDEST	X Target-Co-ordinate
$02B3-02B4	691-692	YDEST	Y " "
$02B5-02B6	693-694	XABS	Absolute value of X
$02B7-02B8	695-696	YABS	" " " Y
$02B9-02BA	697-698	XSGN	Sign of X
$02BB-02BC	699-700	YSGN	Sign of Y
$02BD-02BE	701-702	FCT1	
$02BF-02C0	703-704	FCT2	
$02C1-02C2	705-706	ERRVAL	
$02C3	707	LESSER	

		GREATR	
$02C5	709	ANGSGN	Stat Value of angle
$02C6-02C7	710-711	SINVAL	Sines of angle
$02C8-02C9	712-713	COSVAL	Cosines of angle
$02CA-02CB	714-715	ANGCNT	Temp. Reg. for angle
$02CD	717	BNR	Pointer : Start (PRINT USING)
$02CE	718	ENR	Pointer : End
$02CF	719	DOLR	Flag: Dollar
$02D0	720	FLAG	Flag: Comma
$02D1	721	SWE	Counter
$02D2	722	USGN	Value of exponent
$02D3	723	UEXP	Pointer : exponent
$02D4	724	VN	No of of ch. before dec. point
$02D5	725	CHSN	Flag: Adjustment
$02D6	726	VF	No of ch. before dec. point
$02D7	727	NF	No of ch. after dec. point
$02D8	728	POSP	Flag: +/-(Field)
$02D9	729	FESP	Flag: Exponent (Field)
$02DA	730	ETOF	Switch
$02DB	731	CFORM	Character Counter (Field)
$02DC	732	SNO	Preliminary Character No.

$02DD	733	BLFD	Flag: Space Bar/*
$02DE	734	BEGFD	Pointer to beginning of field
$02DF	735	LFOR	Length of Format String
$02E0	736	ENDFD	Pointer to end of field

$02CC-02CD	716-717	XCENTR	
$02CE-02CF	718-719	YCENTR	
$02D0-02D1	720-721	XDIST1	
$02D2-02D3	722-723	YDIST1	
$02D4-02D5	724-725	XDIST2	
$02D6-02D7	726-727	YDIST2	
$02D8-02D9	728-729	DISEND	
$02DA	730	COLCNT	Column Counter: CHAR Command
$02DB	731	ROWCNT	Row " " "
$02DC	732	STRCNT	For CHAR Command
$02CC-02CD	716-717	XCORD1	Pixel 1, X Co-ordinate
$02CE-02CF	718-719	YCORD1	Pixel 1, Y Co-ordinate
$02D0-02D1	720-721	BOXANG	Rotating Angle-Box
$02D2-02D3	722-723	XCOUNT	
$02D4-02D5	724-725	YCOUNT	
$02D6-02D7	726-727	BXLENG	Length of one side

$02D8-02D9	728-729	XCORD2	Pixel 2, X Co-ordinate
$02DA-02DB	730-731	YCORD2	Pixel 2, Y Co-Ordinate
$02CC-02CD	716-717	XCIRCL	Centre of circle-X co-ordinate
$02CE-02Cf	718-719	YCIRCL	Centre of circle-Y co-ordinate
$02D0-02D1	720-721	XRADUS	X-Radius
$02D2-02D3	722-723	YRADUS	Y-Radius
$02D4-02D5	724-725	ROTANG	Rotating Angle
$02D6-02D7	726-727		
$02D8-02D9	728-729	ANGBEG	Circle-Segment-Angle Start
$02DA-02DB	730-731	ANGEND	" " " " End
$02DC-02DD	732-733	XRCOS	X Radius * Cosine of rotating angle
$02DE-02DF	734-735	YRSIN	Y Radius * Sine of rotating angle
$02E0-02E1	736-737	XRSIN	X Radius * Sine of rotating angle
$02E2-02E3	738-739	YRCOS	Y Radius * Cosine of rotating angle
$02CD	717	KEYLEN	
$02CE	718	KEYNXT	

$02CF	719	STRSZ	Length of String Variables (SHAPE)
$02D0	720	GETTYP	Set Shape-Mode
$02D1	721	STRPTR	Counter for String Position
$02D2	722	OLDBYT	Old Bitmap byte
$02D3	723	NEWBYT	Variable for new string or Bitmap-Byte
$02D4	724		
$02D5-02D6	725-726	XSIZE	Length of Shapes in X-Direction
$02D7-02D8	727-728	YSIZE	" " " Y-Direction
$02D9-02DA	729-730	XSAVE	Temporary memory for X size
$02DB-02DC	731-732	STRADR	Memory for Shape or string descriptor
$02DD	733	BITIDX	Pointer to a Bit in a Byte
$02DE-02E1	734-737	SAVSIZ	Temporary Memory
$02E4	740	CHRPAG	High byte of a Character ROM address for CHAR-Command
$02E5	741	BITCNT	Register for GSHAPE
$02E6	742	SCALEM	Flag: SCALE Mode ($00=Off)
$02E7	743	WIDTH	Flag: Double Pixel Size
$02E8	744	FILFLG	Flag: Colour a rectangle (BOX Command)
$02E9	745	BITMSK	Temporary Memory for Bit Mask

$02EA	746	NUMCNT	Length of String
$02EB	747	TRCFLG	Flag: TRACE Mode ($00=Off)
$02EC	748	T3	Intermediate Mem. for DIRECTORY
$02ED-02EE	749-750	T4	" " " "
$02EF	751	VTEMP3	Temporary Memory for Graphics
$02F0	752	VTEMP4	No. of Graphic Parameters
$02F1	753	VTEMP5	Parameter: $01=relative $00 = absolute
$02F2-02F3	754-755	ADRAY1	Pointer to conversion routine Comma after Integer
$02F4-02F5	756-757	ADRAY2	Pointer to conversion routine Integer after Comma
$02FE-02FF	766-767	BNKVEC	Vector: Function Module (cart.)
$0300-0301	768-769	IERROR	Vector:BASIC Error Mess.($8686)
$0302-0303	770-771	IMAIN	Vector:BASIC Warm Start ($8712)
$0304-0305	772-773	ICRNCH	Vector:BASIC Token gener. $8956
$0306-0307	774-775	IQPLOP	Vector: BASIC Text list ($8B6E)
$0308-0309	776-777	IGONE	Vector BASIC Command exec.$8BD6
$030A-030B	778-779	IEVAL	Vector:BASIC Token Eval. ($9417)
$030C-030D	780-781	IESCLK	Vector:BASIC User Token generation $896A
$030E-030F	782-783	IESCPR	Vector: Create keyword $8B88
$0310-0311	784-785	IESCEX	Vector: Prepare User Token $8C8B

$0312-0313	786-787	ITIME	Vector: Interrupt (Clock)	$CE42
$0314-0315	788-789	CINV	Vector: Hardware Interrupt	$CE0E
$0316-0317	790-791	CBINV	Vector: BRK-Interrupt	
$0318-0319	792-793	IOPEN	Vector:Kernal OPEN Routine	$EF53
$031A-031B	794-795	ICLOSE	Vector:Kernal CLOSE "	$EE5D
$031C-031D	796-797	ICHKIN	Vector:Kernal CHKIN "	$ED18
$031E-031F	798-799	ICKOUT	Vector:Kernal CHKOUT "	$ED60
$0320-0321	800-801	ICLRCH	Vector:Kernal CLRCHN "	$EF0C
$0322-0323	802-803	IBASIN	Vector:Kernal CHRIN "	$EBE8
$0324-0325	804-805	IBSOUT	Vector:Kernal CHROUT "	$EC4B
$0326-0327	806-807	ISTOP	Vector:Kernal STOP "	$F265
$0328-0329	808-809	IGETIN	Vector:Kernal GETIN "	$EBD9
$032A-032B	810-811	ICLALL	Vector:Kernal CLALL "	$EF08
$032C-032D	812-813	USRCMD	Vector:Monitor Break	$F44C
$032E-032F	814-815	ILOAD	Vector:Kernal LOAD Routine	$F04A
$0330-0331	816-817	ISAVE	Vector:Kernal SAVE Routine	$F1A4
$0332-03F2	818-1010	TAPBUF	Tape Buffer	
$0332	818		File type(0=BASIC,1=M/C)	
$0333-0334	819-820		Start Address (low/high)	
$0335-0336	821-822		End Address (low/high)	
$0337-0346	823-838		Program-Name (16 characters)	

$03F3-03F4	1011-1012	WRLEN	Data Counter (Write)
$03F5-03F6	1013-1014	RDCNT	Data Counter (Read)
$03F7-0436	1015-1078	INPQUE	RS-232 Input Buffer (64 Bytes)
$0437-0454	1079-1108	ESTAKL	Tape Error Stack (Low Bytes)
$0455-0472	1109-1138	ESTAKH	" " " (High Bytes)
$0473-0478	1139-1144	CHRGET	Subroutine: Read next Byte from BASIC-Text
$0479-0484	1145-1156	CHRGOT	Read again same Text-Byte
$0485-0493	1157-1171	QNUM	
$0494-04A1	1172-1185	INDSUB	Subroutine for loading from optional bank
$04A2-04A4	1186-1188	ZERO	Numerical constant for BASIC
$04A5-04AF	1189-1199	INDTXT	Text pointer
$04B0-04BA	1200-1210	INDIN1	Index 1
$04BB-04C5	1211-1221	INDIN2	Index 2
$04C6-04D0	1222-1232	INDST1	String 1
$04D1-04DB	1233-1243	INDLOW	Lowtr
$04DC-04E6	1244-1254	INDFMO	Facmo
$04E7	1255	PUFILL	Fill Character by PRINT USING (Standard: Space Bar)
$04E8	1256	PUCOMA	Comma symbol
$04E9	1257	PUDOT	Dot Symbol
$04EA	1258	PUMONY	Currency Character

$04EB-04EE	1259-1262	TMPDES	Temporary memory for INSTR
$04EF	1263	ERRNUM	Last Error Number
$04F0-04F1	1264-1265	ERRLIN	Row no. in which last error occurred ($FFFF=NO error)
$04F2-04F3	1266-1267	TRAPNO	Reference to row no. for ON ERROR GO TO
$04F4	1268	TMPTRP	Temporary register for TRAP
$04F5-04F6	1269-1270	ERRTXT	Intermediate Memory for TRAP
$04F7	1271	OLDSTK	BASIC Text Pointer to last error
$04F8-04F9	1272-1273	TMPTXT	DO-Memory for BASIC Text Pointer
$04FA-04FB	1274-1275	TMPLIN	DO-Memory for row no.
$04FC-04FD	1276-1277	MTIMLO	Low byte of sound 1/2 length
$04FE-04FF	1278-1279	MTIMHI	High byte of sound 1/2 length
$0500	1280	USRPOK	USR Jump Command
$0501-0502	1281-1282	USRADD	USR Address (low/high)
$0503-0507	1283-1287	RNDX	Start Value for RND
$0508	1288	DEJAVU	Flag: Cold or warm start
$0509-0512	1289-1298	LAT	Table of logical file numbers
$0513-051C	1299-1308	FAT	Table of device numbers
$051D-0526	1309-1318	SAT	Table of secondary addresses
$0527-0530	1319-1328	KEYD	Keyboard Buffer (FIFO)
$0531-0532	1329-1330	MEMSTR	Pointer: Start address of

			RAM for operating system
$0533-0534	1331-1332	MSIZ	Pointer:End of RAM Op. system
$0535	1333	TIMOUT	Flag: Timeout (overrun) of (optional) IEC-Bus
$0536	1334	FILEND	1=end of file reached othewise 0
$0537	1335	CTALLY	No. of characters in buffer (for Read and Write)
$0538	1336	CBUFVA	No. of all valid characters in buffer (read only)
$0539	1337	TPTR	Pointer: next ch. in buffer R/W or Read and Write)
$053A	1338	FLTYPE	Type of tape file
$053B	1339	COLOR	Active Attribute Byte (colour, brightness, flashing)
$053C	1340	FLASH	Flag: Character flashes ($00=No)
$053D	1341		Unused
$053E	1342	HIBASE	Video RAM start (Page)
$053F	1343	XMAX	Size of keyboard buffer
$0540	1344	RPTFLG	Flag: Key repeat ($80=all $40=none,$00=only DEL,CRSR,SPC)
$0541	1345	KOUNT	Counting speed for repeat
$0542	1346	DELAY	Counter for repeat delay
$0543	1347	SHFLAG	Flag: Keys SHIFT,CTRL,CBM key
$0544	1348	LSTSHF	Last SHIFT Sample of keyboard

$0545-0546	1349-1350	KEYLOG	Pointer: K.B Decoding Table
$0547	1351	MODE	Flag:SHIFT $80=No,$00=Yes
$0548	1352	AUTODN	Flag:Aut.Scroll.(down)0=On()=Off
$0549	1353	LINTMP	Intermediate mem. in scr.Pr-out
$054A	1354	ROLFLG	" " " "
$054B	1355	FORMAT	Work Mem.of Machine Code Monitor
$054C-054E	1356-1358	MSAL	For assembler
$054F	1359	WRAP	Temporary Memory for assembler
$0550	1360	TMPC	" " " "
$0551	1361	DIFF	" " " "
$0552	1362	PCH	Program-Counter (high)
$0553	1363	PCL	" " (low)
$0554	1364	FLGS	Processor-Flags
$0555	1365	ACC	Accumulata of Processor
$0556	1366	XR	X-Register " "
$0557	1367	YR	Y-Register " "
$0558	1368	SP	Processor Stack Pointer
$0559	1369	INVL	Memory for Machine-Monitor
$055A	1370	INVH	" " " "
$055B	1371	CMPFLG	" " " "
$055C	1372	BAD	Used by various Monitor Routines

$055D	1373	KYNDX	Character counter: Function Key
$055E	1374	KEYIDX	Pointer: Strings function key
$055F-0566	1375-1382	KEYBUF	Length of function key strings
$0567-05E6	1383-1510	PKYBUF	Memory for Function key strings
$05E7	1511	KDATA	Temp. Mem. For data after DMA
$05E8	1512	KDYCMD	Selection, if DMA. Read or Write
$05E9	1513	KDYNUM	DMA device no.
$05EA	1514	KDYPRS	$FF=DMA present, otherwise $00
$05EB	1515	KDYTYP	Temporary Memory for DMA.
$05EC-06EB	1516-1771	SAVRAM	Banking routines page
$05EC-05EF	1516-1519	PAT	Physical Address (Table)
$05F0-05F1	1520-1521	LNGJMP	Long-Jump(Address)
$05F2	1522	FETARG	" " (Accumulator)
$05F3	1523	FETXRG	" " (X-Register)
$05F4	1524	FETSRG	" " (Status-Register)
$05F5-065D	1525-1629	AREA	RAM-Field for Bank-Switching
$065E-06EB	1630-1771	APECH	" " for Speech Synthesizer
$06EC-07AF	1772-1967	STKTOP	Pseudo Stack: BASIC Interpreter
$07B0-07CC	1968-1996	TAWKVA	Tape working values
$07B0	1968	WROUT	Byte to be written on cassette
$07B1	1969	PARITY	Temporary Memory for parity

$07B2	1970	TT1	Temporary Memory
$07B3	1971	TT2	Temporary Memory
$07B5	1973	RDBITS	Local Index for READBYTE-Routine
$07B6	1974	ERRSP	Pointer: Error-Stack
$07B7	1975	FPERRS	No of errors on pass 1
$07B8-07B9	1976-1977	DSAMP1	Time Constant
$07BA-07BB	1978-1979	DSAMP2	" "
$07BC-07BD	1980-1981	ZCELL	" "
$07BE	1982	SRECOV	Stack Pointer for STOP-Key
$07BF	1983	DRECOV	" " " DROP-Key
$07C0-07C3	1984-1987	TRSAVE	Parameters sent after RDBLOK
$07C4	1988	RDETMP	Temporary Memory for RDBLOK
$07C5	1989	LDRSCN	
$07C6	1990	CDERRM	No of errors on RD Countdown
$07C7	1991	VSAVE	Temporary Memory for VERIFY
$07C8-07CB	1992-1995	T1PIPE	Temporary Memory for T1
$07CC	1996	ENEXT	Read Error
$07CD	1997	UOUTQ	User Char. to be sent (RS-232)
$07CE	1998	UOUTFG	(0=empty, 1=full)
$07CF	1999	SOUTQ	System Character to be sent
$07D0	2000	SOUNFG	(0=empty, 1=full)

$07D1	2001	INQFPT	Pointer to start of input buffer
$07D2	2002	INQRPT	" " end of " "
$07D3	2003	INQCNT	No of Char. in Input Buffer
$07D4	2004	ASTAT	Temporary status of ACIA
$07D5	2005	AINTMP	Temporary memory for INPUT
$07D6	2006	ALSTOP	Flag: Local Pause
$07D7	2007	ARSTOP	Flag: Remote Control Pause
$07D8	2008	APRES	Flag: Marks presence of ACIA
$07D9-07E4	2009-2020	KLUDES	Indirect Routine (downloaded)
$07E5	2021	SCBOT	Screen window: bottom edge
$07E6	2022	SCTOP	" " top edge
$07E7	2023	SCLF	" " left edge
$07E8	2024	SCRT	" " right edge
$07E9	2025	SCRDIS	
$07EA	2026	INSFL	Flag: automatic inserting
$07EB	2027	LSTCH	Last printed character
$07EC	2028	LOGSCR	Memory for Screen Administration
$07ED	2029	TCOLOR	Temp.reg.for color on INST & DEL
$07EE-07F1	2030-2033	BITABL	Row Link Table for Screen
$07F2	2034	SAREG	Save Accumulator on SYS-Command
$07F3	2035	SXREG	Save X-Register " " "

$07F4	2036	SYREG	Save Y-Register " " "
$07F5	2037	SPREG	Save SP-Register " " "
$07F6	2038	LSTX	Currently pressed key: CHR$(n),
$07F7	2039	STPDSB	Flag: CTRL-S (0=open, 6=closed)
$07F8	2040	RAMROM	RAM/ROM-Switch over to machine code monitor ($00=ROM, $80=RAM)
$07F9	2041	COLSW	RAM/ROM-Switch over for colour Brightness-Table:$00=ROM,$80=RAM
$07FA	2042	FFRMSK	ROM-Mask for divided screen
$07FB	2043	VMBMSK	Video RAM Mask for " "
$07FC	2044	LSEM	Motor control tape recorder
$07FD	2045	PALCNT	Auxiliary counter for clock on PAL-System
$07FE-07FF	2046-2047		Unused

BEFORE CALLING HI-RES. GRAPHICS

$0800-0BE7	2048-3047	TEDATR	Colour-RAM (Text Mode)
$0C00-0FE7	3072-4071	TEDSCN	Video-RAM (Text Mode)
$1000-3FFF	4096-16383	BASBGN	Basic RAM----------C16
$1000-7FFF	4096-32767	BASBGN	BASIC RAM----------C16+16K
$1000-FCFF	4096-64767	BASBGN	BASIC RAM-------C16+64K or PLUS4

AFTER CALLING HI-RES GRAPHICS

$1800-1BE7 6144-7143 TEDATR Luminscence Table (Graphics Mode)

$1C00-1FE7 7168-8167 TEDSCN Colour Table (Graphics Mode)

$2000-3F3F 8192-16191 GRBASE Bitmap Graphics Screen

$1000-17FF 4096-6143 BASBGN BASIC RAM ---------- C16

$4000-7FFF 16384-32767 BASBGN BASIC RAM ---------- C16+16K

$4000-FCFF 16384-64767 BASBGN BASIC RAM --------- PLUS4:C16+64K

PLUS 4 ONLY

$8000-CFFF 32768-53247 BASIC Interpreter

$D000-D3FF 53248-54271 Character ROM (Capitals/Graphics Ch. set

$D400-D7FF 54272-55295 Character ROM (Small /Capital Character

$D800-FBFF 55296-64511 Operating System

$FC00-FCFF 64512-64767 ROM Banking Routines

$FD00-FD0F 64768-64783 ACIA (Only PLUS/4)

$FD10-FD1F 64784-64799 6529 Parallel Port (Only PLUS/4)

$FDD0-FDDF 64976-64991 Module Bank Port

$FE00-FEFF 65024-65279 DMA Disk System

4.6 TED CHIP ADDRESSES ($FF00-$FF3F)

REG.	ADDRESS HEX.	ADDRESS DEC.	BITS	DESCRIPTION
0	$FF00	65280		Timer 1, Reload Low
1	$FF01	65281		Timer 1, Reload High
2	$FF02	65282		Timer 2, Low
3	$FF03	65283		Timer 2, High
4	$FF04	65284		Timer 3, Low
5	$FF05	65285		Timer 3, High
6	$FF06	65286	0-2	Vertical Scroll Position (Y)
			3	Selection of 24/25 rows (1=25)
			4	Switch screen off
			5	Switch Bitmap Mode (1=ON)
			6	Switch Ext. Colour Mode(1=ON)
			7	Testbit (always 0)
7	$FF07	65287	0-2	Horizontal Scroll Position (X)
			3	Choice of 38/40 columns (1=40)
			4	Switch Multicolour Mode on (1=on)
			5	Switch Freeze Mode on (1=on)
			6	PAL/NTSC-Mode (0=PAL, 1=NTSC)
			7	RVS-Video (0=Hardware, 1=Software)
8	$FF08	65288		Keyboard Matrix

9	$FF09	65289		Interrupt-Sources
			0	Not used
			1	Raster Interrupt
			2	(Light Pen, not possible with C-16)
			3	Timer 1 Interrupt
			4	Timer 2 Interrupt
			5	Not used
			6	Timer 3 Interrupt
			7	Interrupt Bit
10	FF0A	65290		Interrupt Masking
			0	Bit 8 of Raster Comparison (Reg. 11)
			1	Raster-Interrupt
			2	(Light Pen, not possible with C-16
			3	Timer 1 Interrupt
			4	Timer 2 Interrupt
			5	Not used
			6	Timer 3 Interrupt
			7	Not used
11	FF0B	65291		Raster Comparison (Bit 0-7)
12	FF0C	65292	0-1	Bit 8-9 of Cursor Position (Reg. 13)
			2-7	Not Used
13	FF0D	65293		Hardware Cursor Position (Bit 0-7)
14	FF0E	65294		Frequency Voice 1 (Bit 0-7)
15	FF0F	65295		" " 2 (Bit 0-7)
16	FF10	65296	0-1	Frequency Voice 2 (Bit 8-9)
			2-7	Not used

17	FF11	65297	0-3	Volume (0=off, 15=loud)
			4	Switch voice 1 on (1=on)
			5	Switch voice 2 rectangular on (1=on)
			6	Switch voice 2 noise on (1=on)
			7	Sound Reload Bit
18	FF12	65298	0-1	Bit 8-9 Frequency Voice 1 (Reg. 14)
			2	RAM/ROM Bank (0=RAM, 1=ROM)
			3-5	Address of Bitmap RAM (Bit 13-15)
			6-	Not used
19	FF13	6529	0	ROM-Bank Status Bit (Read only)
			1	Force Single Clock Bit (1=prohibits double time frequency)
			2-7	Address of character set (Bit 10-15)
20	FF14	65300	0-2	Not used
			3-7	Address of Video-RAM (Bit 11-15)
21	FF15	6530	0-3	Background 0 Colour
			4-6	Background 0 Luminiscence
			7	Not used
22	FF16	65302	0-3	Background 1 Colour
			4-6	" " Luminiscence
			7	Not used
23	FF17	65303	0-3	Background 1 colour
			4-6	" " Luminiscence
			7	Not used
24	FF18	6530	0-3	Background 3 Colour
			4-6	" " Luminiscence
			7	Not used
25	FF19	65305	0-3	Frame Colour
			4-6	" Luminiscence
			7	Not used

26	FF1A	6530	0-1	Bit 8-9 of Bit Map Reload (Reg.27)
			2-7	Not used
27	FF1B	65307		Bit Map Reload of Character Position (Bit 0-7)
28	FF1C	65308	0	Bit 8 of Raster-Row (Reg. 29)
			1-7	Not used
29	FF1D	65309		Current Raster Row (Bit 0-7)
30	FF1E	65310		Current raster Column (Bit 1-8)
31	FF1F	65311	0-2	Vertical Sub Address
			3-6	Flash Rate
			7	Not used
62	FF3E	65342		ROM-Select (Write only)
63	FF3F	65343		RAM-Select (Write only)

4.7 KERNAL JUMP TABLE ($FF49-$FFFF)

ADDRESS (HEX)	ADDRESS (DEC)	CODE	NAME	DESCRIPTION
$FF49	65353	JMP $B7C2		Define FUNCTION key no. after $76,addr. after $22-23, length in accumulator)
$FF4C	65356	JMP $DC49		Print
$FF4F	65359	JMP $FBD8		Print message
$FF52	65362	JMP $F445		Call M/C Monitor
$FF55-	65365			Not used
$FF7E	65406			Not used
$FF7F	65407	$2A		Not used
$FF80	65408	$84		" "
$FF81	65409	JMP $D84E	CINT	Initialise Editor
$FF84	65412	JMP $F30B	IOINIT	Initialise I/O
$FF87	65415	JMP $F352	RAMTAS	Initialise RAM, open cassette buffer, screen to $0C00
$FF8A	65418	JMP $F2CE	RESTOR	Restore vectors
$FF8D	65421	JMP $F2D3	VECTOR	Vector RAM
$FF90	65424	JMP $F41A	SETMSG	Control KERNAL Messages
$FF93	65427	JMP $EE4D	SECOND	Send sec. addr. to LISTEN

$FF96	65430	JMP $EE1A	TKSA	Send sec. addr. to TALK	
$FF99	65433	JMP $F427	MEMTOP	Set/read top RAM Pointer	
$FF9C	65436	JMP $F436	MEMBOT	Set/read bottom RAM Pointer	
$FF9F	65439	JMP $DB11	SCNKEY	Scan keyboard	
$FFA2	65442	JMP $F423	SETTMO	Set Timeout for (optional) IEC-Bus	
$FFA5	65445	JMP $EC8B	ACPTR	Input byte from serial port	
$FFA8	65448	JMP $ECDF	CIOUT	Output byte via serial port	
$FFAB	65451	JMP $EF3B	UNTLK	Command serial bus to UNTALK	
$FFAE	65454	JMP $EF23	UNLSN	Com. serial bus to UNLISTEN	
$FFB1	65457	JMP $EE2C	LISTEN	Com.all dev. on bus to LISTEN	
$FFB4	65460	JMP $EDFA	TALK	Com. dev. on serial bus to TALK	
$FFB7	65463	JMP $F41C	READST	Read I/O Status Word	
$FFBA	65466	JMP $F413	SETLFS	Set Logical, Primary and Secondary Address	
$FFBD	65469	JMP $F40C	SETNAM	Determine file names	
$FFC0	65472	JMP ($0318)	OPEN	Open a logical file	$EF53
$FFC3	65475	JMP ($031A)	CLOSE	Close a logical file	$EE5D
$FFC6	65478	JMP ($031C)	CHKIN	Open channel for input	$ED18
$FFC9	65481	JMP ($031E)	CHKOUT	Open channel for output	$ED60
$FFCC	65484	JMP ($0320)	CLRCHN	Close I/O channels	$EF0C
$FFCF	65487	JMP ($0322)	CHRIN	Input Character	$EBE8
$FFD2	65490	JMP ($0324)	CHROUT	Output Character	$EC4B

$FFD5	65493	JMP $F043	LOAD	Load from peripheral device	
$FFD8	65496	JMP $F194	SAVE	Store on peripheral device	
$FFDB	65499	JMP $CF2D	SETTIM	Set time	
$FFDE	65502	JMP $CF26	RDTIM	Read time	
$FFE1	65505	JMP ($0326)	STOP	Scan STOP Key	$F265
$FFE4	65508	JMP ($0328)	GETIN	Read ch. from K.B. buffer	$EBD9
$FFE7	65511	JMP ($032A)	CLALL	Close all channels & logical files	$EF08
$FFEA	65514	JMP $CEF0	UDTIM	Increment Time	
$FFED	65517	JMP $D834	SCREEN	Identify X, Y Screen Set-Up	
$FFF0	65520	JMP $D839	PLOT	Read/set X, Y Cursor Positiong	
$FFF3	65523	JMP $FC19	IOBASE	Base address-Reports back on I/O devices.	
$FFF6	65526	STA $FF3E		Switch-on ROM	
$FFF9	65529	JMP $F2A4		Jump to Reset Routine	
$FFFC	65532	$FFF6		Processor Reset	
$FFFE	65534	$FCB3		Processor Interrupt	

4.8 COMPARISON TABLE C-64 AND C-16

In many newspapers, programs for the CBM 64 are published which could also be of interest to the C-16 owner. While the programs to some extent try to teach the C-64 BASIC Commands, which the C-16 has built-in anyway (e.g. Graphics-Commands) and are therefore uninteresting for you, there are surely some programs which you would also like to have on your C-16. To facilitate the re-writing, we have compiled a comparison chart of all identical addresses. Since many C-64 programs have numerous PEEK and POKE commands, you can look for the corrosponding C-16 Address in this table. You cannot, of course, re-write everything (e.g. programs with Sprites) but what is comparable is listed in the following table. The explanation of the meaning of the address is very short; a more detailed description can be found in the previous chapter.

ADDRESS C64 hex	ADDRESS C64 dec	ADDRESS C16 hex	ADDRESS C16 dec	LABEL	MEANING
$0000	0	$0000	0	D6510	Data Direction Register
$0001	1	$0001	1	R6510	I/O-Port
$0003	3	$02F2	754	ADRAY1	Pointer to Conv. GK>IN
$0005	5	$02F4	756	ADRAY2	" " " IN>GK
$0007	7	$0007	7	CHARAC	Search Character
$0008	8	$0008	8	ENDCHR	FL:Search for inv. COMMA
$0009	9	$0009	9	TRMPOS	Screen col. from 1. TAB
$000A	10	$000A	10	VERCK	FL:0=LOAD, 1=VERIFY

$000B	11	$000B	11	COUNT	Input buffer pointer
$000C	12	$000C	12	DIMFLG	FL:Standard field dimen.
$000D	13	$000D	13	VALTYP	Data type: $FF=String
$000E	14	$000E	14	INTFLG	Data type: $80=Integer
$0010	16	$0010	16	SUBFLG	FL:User Function
$0011	17	$0011	17	INPFLG	FL:$00=INPUT, $40=GET
$0012	18	$0012	18	TANSGN	FL:TAN sign
$0013	19	$0013	19	CHANNL	FL:INPUT Comment
$0014-	20-	$0014-	20-	LINNUM	Total num. value
$0016	22	$0016	22	TEMPPT	PTR:Temp. string stack.
$0017-	23	$0017-	23-	LASTPT	Last String Address
$0019-	25-	$0019-	25-	TEMPST	Stack for temp.String
$0022-	34-	$0022-	34-	INDEX	Field for aux. pointer
$0026-	38-	$0026-	38-	RESHO	Field for multiplier.
$002B-	43-	$002B-	43-	TXTTAB	PTR:Start BASIC text
$002D-	45-	$002D-	45-	VARTAB	PTR:Start BASIC Variable.
$002F-	47-	$002F-	47-	ARYTAB	PTR:Start BASIC Arrays.
$0031-	49-	$0031-	49-	STREND	PTR:End BASIC Strings(+1)
$0033-	51-	$0033-	51-	FRETOP	PTR:Start of Strings
$0035-	53-	$0035-	53-	FRESPC	PTR:Aux. Strings
$0037-	55-	$0037-	55-	MEMSIZ	PTR:Top BASIC Address

$0039-	57-	$0039-	57-	CURLIN	Current BASIC Line
$003B-	59-	$0259-	601-	OLDLIN	Previous BASIC Line
$003D-	61-	$025B-	602-	OLDTXT	PTR:BASIC for CONT
$003F-	63-	$003F-	63-	DATLIN	Current DATA Line
$0041-	65-	$0041-	65-	DATPTR	PTR:Current DATA Address
$0043	67-	$0043	67-	INPPTR	V: INPUT Routine
$0045	69-	$0045-	69-	VARNAM	Curr. BASIC Var. Name
$0047-	71-	$0047-	71-	VARPNT	Address of curr. vari.
$0049-	73-	$0049-	73-	FORPNT	Var. ch.for FOR/NEXT
$004B-	75-	$004B-	75-	OPPTR	Int. ch.for BASIC
$0061	97	$0061	97	FACEXP	Floating Point 1 Expo.
$0062-	98-	$0062-	98-	FACHO	" " 1 Mant.
$0066	102	$0066	102	FACSGN	" " 1 sign
$0067	103	$0067	103	SGNFLG	PTR:Polynom. Evaluation
$0068	104	$0068	104	BITS	Floating Point 1 overfl.
$0069	105	$0069	105	ARGEXP	" " 2 Expo.
$006A-	106-	$006A-	106-	ARGHO	" " 2 Mant.
$006E	110	$006E	110	ARGSGN	" " 2 sign
$006F	111	$006F	111	ARISGN	Sign Comp. Acc 1 & 2
$0070	112	$0070	112	FACOV	Floating pt. 1 rounding
$0071-	113-	$0071-	113-	FBUFPT	PTR:Cassette Buffer

$0073-	115-	$0473-	1139-	CHRGET	Read sub routine
$0079	121	$0479	1145	CHRGOT	Repeat read
$007A-	122-	$003B-	59-	TXTPTR	PTR:Current Byte of Text
$008B-	139-	$0503-	1283-	RNDX	Start value for RND
$0090	144	$0090	144	STATUS	Kernal I/O Status Word
$0091	145	$0091	145	STKEY	PTR:STOP/RVS-Key
$0093	147	$0093	147	VERFCK	FL:0=LOAD, 1=VERIFY
$0094	148	$0094	148	C3PO	FL:Ser. Bus, Character
$0095	149	$0095	149	BSOUR	Character in buffer
$0098	152	$0097	151	LDTND	No. of open files
$0099	153	$0098	152	DFLTN	Input device(Defaul=3)
$009A	154	$0099	153	DFLTO	Output dev. (Default=0)
$009D	157	$009A	154	MSGFLG	FL:$80=Direct, $00=Pr.
$00A0-	160-	$00A3-	163-	TIME	Clock (approx. 1/60)
$00AC-	172-	$009B-	155-	SAL	PTR:Cass.buf/Scn. scroll
$00AE-	174-	$009D-	157-	EAL end	PTR:Cassette end/Program
$00B7	183	$00AB	171	FNLEN	Lnth. of curr. file name
$00B8	184	$00AC	172	LA	Logical Data file No.
$00B9	185	$00AD	173	SA	Current Second. Address
$00BA	186	$00AE	174	FA	Current device no.
$00BB-	187-	$00AF-	175-	FNADR	PTR:Current file Name

$00C1-	193-	$00B2-	178-	STAL	I/O Start Address
$00C3-	195-	$00B4-	180-	MEMUSS	Basic Loading Address
$00C5	197	$07F6	2038	LSTX	Currently pressed key
$00C6	198	$00EF	239	NDX	No of ch. in K.B. buffer
$00C7	199	$00C2	194	RVS	FL:RVS Character (1=Yes)
$00C8	200	$00C3	195	INDX	PTR:End of logical line
$00C9-	201	$00C4-	196-	LSXP	Cursor X/Y for Input
$00CB	203	$00C6	198	SFDX	FL:Pressed key
$00D	208	$00C7	199	CRSW	FL:Input INPUT or GET
$00D1-	209-	$00C8-	200-	PNT	PNT:Current Screen Char.
$00D3	211	$00CA	202	PNTR	Cursor column in current line
$00D4	212	$00CB	203	QTSW	FL:Inverted comma mode
$00D6	214	$00CD	205	TBLX	Cursor line
$00D8	216	$00CF	207	INSRT	FL:INST Mode
$00F3-	243-	$00EA-	234-	USER	PTR:Current Colour RAM
$00F5-	245-	$00EC-	236-	KEYTAB	V: Keyboard Decod.
$0100	256-	$0100-	256-		Processor Stack
$0200	512-	$0200-	512-	BUF	System Input Buffer
$0259-	601-	$0509-	1289-	LAT	Table of log. file no.
$0263-	611-	$0513-	1299-	FAT	Table of device nos.

$026D-	621-	$051D-	1309-	SAT	Table of sec. addr.
$0277-	631-	$0527-	1319-	KEYD	Keyboard Buffer (FIFO)
$0281-	641-	$0531-	1329-	MEMSTR	PTR:Start Address RAM
$0283-	643-	$0533-	1331-	MSIZ	PTR:End Address RAM
$0285	645	$0535	1333	TIMOUT	Fl:Time overrun IEC
$0286	646	$053B	1339	COLOR	Character colour
$0288	648	$053E	1342	HIBASE	Video RAM Start, Page
$0289	649	$053F	1343	XMAX	Size of Keyboard Buffer
$028A	650	$0540	1344	RPTFLG	FL:Key repeat
$028B	651	$0541	1345	KOUNT	Counting speed for repeat
$028C	652	$0542	1346	DELAY	Counter for repeat delay
$028D	653	$0543	1347	SHFLAG	FL:Keys SHIFT, CTRL
$028E	654	$0544	1348	LSTSHF	Last SHIFT Sample
$028F-	655	$0545-	1349-	KEYLOG	Pointer to key decod.
$0291	657	$0547	1351	MODE	Fl:$80=SHIFT ineffective
$0292	658	$0548	1352	AUTODN	FL:Automatic Scrolling
$0300-	768-	$0300	768-	IERROR	V: BASIC-Error Message
$0302-	770-	$0302-	770-	IMAIN	V: BASIC Warm start
$0304-	772-	$0304-	772-	ICRNCH	V: BASIC Token gener.
$0306-	774-	$0306-	774-	IQPLOP	V: BASIC Text listen
$0308	776-	$0308-	776-	IGONE	V: Execute BASIC-Command

$030A-	778-	$030A-	778-	IEVAL	V: BASIC Token Evaluation
$030C	780	$07F2	2034	SAREG	Acc. for SYS-Command
$030D	781	$07F3	2035	SXREG	X-Reg. for SYS-Command
$030E	782	$07F4	2036	SYREG	Y-Reg. for SYS Command
$030F	783	$07F5	2037	SPREG	SP-Reg. for SYS command
$0310	784	$0500	1280	USRPOK	USR Jump
$0311-	785-	$0501-	1281-	USRADD	USR-Address (low/high)
$0314-	788-	$0314-	788-	CINV	V: Hardware-Interrupt
$0316-	790-	$0316-	790-	CBINV	V: BRK-Interupt
$031A-	794-	$0318-	792-	IOPEN	V: Kernal OPEN
$031C-	796-	$031A-	794-	ICLOSE	V: KERNAL CLOSE
$031E-	798-	$031C-	796-	ICHKIN	V: KERNAL CHKIN
$0320-	800-	$031E-	798-	ICKOUT	V: " CHKOUT
$0322-	802-	$0320-	800-	ICLRCH	V: " CLRCHN
$0324-	804-	$0322-	802-	IBASIN	V: " CHRIN
$0326-	806-	$0324-	804-	IBASOUT	V: " CHROUT
$0328-	808-	$0326-	806-	ISTOP	V: " STOP
$032A-	810-	$0328-	808-	IGETIN	V: " GETIN
$032C	812-	$032A-	810-	ICLALL	V: " CLALL
$032E-	814-	$032C-	812-	USRCMD	User IRQ (Monitor)
$0330-	816-	$032E-	814-	ILOAD	V: Kernal-LOAD

$03332-	818-	$0330-	816-	ISAVE	V: Kernal-SAVE
$033C-	828-	$0332-	818-	TAPBUF	Cassette-Buffer
$0400-	1024-	$0C00-	3076	VICSCN	Video-RAM
$0800-	2048-	$1000-	4096-	BASBGN	BASIC-RAM
$D011	53265	$FF06	65286		Bit 7 is different!
$D012	53266	$FF0B	65291		Raster Interrupt
$D016	53270	$FF07	65287		Bits 5-7 different!
$D020	53280	$FF19	65305		Frame Colour
$D021	53281	$FF15	65301		Background 0 colour
$D022	53282	$FF16	65302		" 1 "
$D023	53283	$FF17	65303		" 2 "
$D024	53284	$FF18	65304		" 3 "
$D800	55296	$0800	2048	COLSCN	Colour-RAM

MEMORY MAP OF THE CBM 64

ADDRESS (hex)	ADDRESS (dec)	DESCRIPTION
$0000-00FF	0- 255	Zero-Page
$0100-01FF	256- 511	Processor-Stack
$0200-03FF	512- 1023	Different Variables,Cassette Buffer
$0400-07E7	1024- 2023	Video-RAM
$0800-9FFF	2048-40959	BASIC-RAM
$A000-BFFF	40960-49151	BASIC-Interpreter
$C000-CFFF	49152-53247	Free RAM (for sub-routines etc.)
$D000-D3FF	53248-54271	Video-Controller (Reg.up to $D02E)
$D400-D7FF	54272-55295	Sound-Controller (Reg.up to $D41C)
$D800-DBE7	55296-56295	Colour-RAM
$DC00-DCFF	56320-56575	CIA #1 : Keyboard, Joystick, Timer
$DD00-DDFF	56576-56831	CIA #2 : Serial Bus, RS-232, Timer
$DE00-DFFF	56832-57343	Not used; free for I/O-Extensions
$E000-FFFF	57344-65535	Kernal-ROM

5. UTILITIES

5.1 RANDOM VALUE GENERATOR IN MACHINE CODE

The following example demonstrates how to create random values in machine code. This is of special interest to games programmers, who frequently needs random values. In our example, a throw with two dice is simulated. When you press the space bar the dice are played and the result of our throw will be shown on the top of the screen. As random start value, the raster beam is used.

```
>1000 00 0B 10 00 00 9E 34 31 :r......4
>1008 31 32 00 00 00 00 00 00 :r12.....
>1010 A9 93 20 D2 FF 20 46 10 :r). r? f
>1018 A9 0F 8D 11 08 8D 13 08 :r)......
>1020 20 E4 FF C9 20 D0 F9 20 :r $?i p9
>1028 35 10 8D 11 0C 20 35 10 :r5.... 5
>1030 8D 13 0C D0 EB 20 50 10 :r...p+ p
>1038 F0 FB C9 07 B0 F7 C9 07 :r0;i.07i
>1040 B0 F3 18 69 30 60 AD 0B :r03.)0 -
>1048 FF A2 04 95 D0 CA 10 FB :r?"..pj.
>1050 A9 E0 25 D4 09 20 85 D4 :r) %t. .
>1058 18 A2 04 B5 D0 75 D0 95 :r.".5p5p
>1060 D5 CA 10 F7 18 A2 04 B5 :ruj.7.".
>1068 D0 75 D5 95 D0 CA 10 F7 :rp5u.pj.
>1070 18 A2 02 B5 D0 75 D7 95 :r.".5p5w
>1078 D0 CA 10 F7 A5 D0 60 AA :rpj.7%p
```

```
. 1010  A9 93    LDA #$93
. 1012  20 D2 FF JSR $FFD2
. 1015  20 46 10 JSR $1046
. 1018  A9 0F    LDA #$0F
. 101A  8D 11 08 STA $0811
. 101D  8D 13 08 STA $0813
. 1020  20 E4 FF JSR $FFE4
. 1023  C9 20    CMP #$20
. 1025  D0 F9    BNE $1020
. 1027  20 35 10 JSR $1035
```

```
. 102A  8D 11 0C STA $0C11
. 102D  20 35 10 JSR $1035
. 1030  8D 13 0C STA $0C13
. 1033  D0 EB    BNE $1020
. 1035  20 50 10 JSR $1050
. 1038  F0 FB    BEQ $1035
. 103A  C9 07    CMP #$07
. 103C  B0 F7    BCS $1035
. 103E  C9 07    CMP #$07
. 1040  B0 F3    BCS $1035
. 1042  18       CLC
. 1043  69 30    ADC #$30
. 1045  60       RTS
. 1046  AD 0B FF LDA $FF0B
. 1049  A2 04    LDX #$04
. 104B  95 D0    STA $D0,X
. 104D  CA       DEX
. 104E  10 FB    BPL $104B
. 1050  A9 E0    LDA #$E0
. 1052  25 D4    AND $D4
. 1054  09 20    ORA #$20
. 1056  85 D4    STA $D4
. 1058  18       CLC
. 1059  A2 04    LDX #$04
. 105B  B5 D0    LDA $D0,X
. 105D  75 D0    ADC $D0,X
. 105F  95 D5    STA $D5,X
. 1061  CA       DEX
. 1062  10 F7    BPL $105B
. 1064  18       CLC
. 1065  A2 04    LDX #$04
. 1067  B5 D0    LDA $D0,X
. 1069  75 D5    ADC $D5,X
. 106B  95 D0    STA $D0,X
. 106D  CA       DEX
. 106E  10 F7    BPL $1067
. 1070  18       CLC
. 1071  A2 02    LDX #$02
. 1073  B5 D0    LDA $D0,X
. 1075  75 D7    ADC $D7,X
. 1077  95 D0    STA $D0,X
. 1079  CA       DEX
```

```
. 107A  10 F7     BPL $1073
. 107C  A5 D0     LDA $D0
. 107E  60        RTS
```

5.2 JOYSTICK SCAN IN MACHINE CODE

The following routine scans a joystick in machine code. We have already used this program several times in the Graphics Chapter as sub routine. It can, however, also be integrated in your own program.

```
>1000 00 0C 10 00 00 9E 34 31 :r......41
>1008 31 32 00 00 00 00 00 00 :r12......
>1010 A9 FD 8D 08 FF AD 08 FF :r)=..?-.?
>1018 A0 00 A2 00 4A B0 01 88 :r ."."j0..
>1020 4A B0 01 C8 4A B0 01 CA :rj0.hj0.j
>1028 4A B0 01 E8 86 D0 84 D1 :rj0.(.p.q
>1030 29 08 60 AA AA AA AA AA :r). *****

. 1010  a9 FD    LDA #$FD
. 1012  8D 08 FF STA $FF08
. 1015  AD 08 FF LDA $FF08
. 1018  A0 00    LDY #$00
. 101A  A2 00    LDX #$00
. 101C  4A       LSR
. 101D  B0 01    BCS $1020
. 101F  88       DEY
. 1020  4A       LSR
. 1021  B0 01    BCS $1024
. 1023  C8       INY
. 1024  4A       LSR
. 1025  B0 01    BCS $1028
. 1027  CA       DEX
. 1028  4A       LSR
. 1029  B0 01    BCS $102C
. 102B  E8       INX
. 102C  86 D0    STX $D0
. 102E  84 D1    STY $D1
. 1030  29 08    AND #$08
. 1032  60       RTS
```

5.3 TURBO MODE FOR THE C 16

For all those, who think the C 16 is too slow, we can show you a trick with which it will become about 30% faster. Since we live in a Turbo age, we simply call it the Turbo Mode. The trick is to switch the screen off. Of course this is impractical if you want to create Graphics and see them at the same time, but not if you have mathematical problems to solve, where you don't always have to look at the screen. The speed advantage arises from the fact that the Video-Chip TED doesn't slow down the Micro Processor anymore once the screen is switched off. A small demo program, first in Turbo Mode and then in the normal mode, will show you how it works and how much time you can save.

```
100 POKE 65286,PEEK(65286) AND 239
110 GOSUB 150
120 POKE 65286,PEEK(65286) OR 16
130 GOSUB 150
140 END
150 T=TI
160 FOR I=1 TO 1000
170 : B=B+1
180 NEXT I
190 PRINT USING "##.##";(TI-T)/60
200 RETURN
```

5.4 OLD (RESTORING A PROGRAM AFTER NEW)

If you accidently type NEW or press the RESET button without having saved the program beforehand,all is not lost. We will show you how you can retreive the program using OLD Program. After NEW or a Reset, you only have to type SYS 1630 and your program is back again. The machine program is stored in an area which is not erased on Reset. If you load the OLD Program before you start programming, it will then be available any time you need it.

PROCEDURE

Type in the following BASIC Program and store it on a cassette or disc before you test it. Enter the monitor to save theprogram.

```
S"OLD",01,065E,06C9----Cass.
S"OLD",08,065E,06C9----Disc
```

When you run it and there is no error then it is ready for use. You can try it on the Basic programme you just wrote. Type NEW and then: SYS 1630. It should now be back again (try LIST). The only thing which might not be correct is the first line number.

To load the program, enter monitor and load the program with L command.

The Basic program is no longer required as it has been saved as M/C program.

```
100 FOR I=1630 to 1630+106
110 : READ A
120 : POKE I,A
130 : C=C+A
140 NEXT I
150 IF C<>10560 THEN PRINT "ERROR" : STOP
160 DATA 165,43,133,3,165,44,133,4,160,4,177,3,240,3,200,208
170 DATA 249,200,152,32,189,6,160,0,165,3,145,43,200,165,4,145
180 DATA 43,200,169,0,145,43,200,145,43,165,43,133,3,165,44,133
```

```
190 DATA 4,160,0,169,4,32,189,6,177,3,240,7,32,187,6,169
200 DATA 0,240,245,32,187,6,177,3,240,4,169,0,240,234,32,187
210 DATA 6,32,187,6,165,3,133,45,165,4,133,46,96,169,1,24
220 DATA 101,3,133,3,169,0,101,4,133,4,96
```

5.5 MERGE (LINKING OF PROGRAMS)

Although the C-16's BASIC is very comprehensive and leaves little to be desired, there is unfortunately one command missing which is very practical: MERGE. This allows linking of two BASIC programs. If you have a library of sub routines, you can always load these and link them to your current program using MERGE routine.

PROCEDURE

Type in the following program and save it on a cassette or disc by first entering the MONITOR. Use the following instructions.

```
S"MERGE",01,06C9,06E8-----Cass.
S"MERGE",08,06C9,06E8-----Disc
```

Type RUN. If there is an error message, check your listing. OLD is stored in the part of memory which is not effected by RESET.

To link two programmes

1. Load MERGE

2. Load your first program from Cassette or Disk.

3. Type: SYS 1737

4. Load your second program.

5. Type: SYS 1759

Both your programs are now linked. You may have to re-number the new program. Avoid using low line numbers in your sub routines. It's best to give your sub-routines high line numbers (e.g. 62000, 63000).

```
100 FOR I=1737 TO 1737+30
110 : READ A
120 : POKE I,A
130 : C=C+A
140 NEXT I
150 IF C<>3786 THEN PRINT "ERROR":STOP
160 DATA 165,43,133,232,165,44,133,233,56,165,45,233,2,133,43,165
170 DATA 46,233,0,133,44,96,165,232,133,43,165,233,133,44,96
```

5.6 VARLIST (LIST OF ALL USED VARIABLES)

If you have created a program in BASIC and lost track of the variables you have already used, then this auxiliary program is useful. It gives a list of all variables so far used and states their type: FLOATING POINT (FLO), INTEGER (INT) or STRING (STR).

PROCEDURE

Type in the BASIC Program and store it. If there is no error, you can try it out. Type: SYS 16100 and the variables I, A and C used in the BASIC Loader will be printed.

```
100 POKE 55,227 : POKE 56,62 : CLR
110 FOR I=16100 TO 16100+273
120 : READ A
130 : POKE I,A
140 : C=C+A
150 NEXT I
160 IF C<>30678 THEN PRINT "ERROR":STOP
170 DATA 165,45,133,2,165,46,133,3,165,2,197,47,208,6,165,3
180 DATA 197,48,240,40,160,0,177,2,141,227,63,201,128,176,30,200
190 DATA 177,2,201,128,176,65,32,106,63,32,211,63,169,7,24,101
200 DATA 2,133,2,169,0,101,3,133,3,76,236,62,96,56,233,128
210 DATA 141,227,63,200,177,2,201,128,240,24,56,233,128,141,228,63
220 DATA 32,134,63,32,147,63,174,238,63,32,164,63,32,205,63,76
230 DATA 13,63,169,32,76,49,63,240,24,56,233,128,141,228,63,32
240 DATA 134,63,32,147,63,174,240,63,32,164,63,32,205,63,76,13
250 DATA 63,169,32,76,80,63,201,0,240,19,141,228,63,32,134,63
260 DATA 32,147,63,174,239,63,32,164,63,32,205,63,96,169,32,76
270 DATA 110,63,173,227,63,32,183,63,173,228,63,32,183,63,96,162
280 DATA 15,142,241,63,169,32,32,183,63,174,241,63,202,208,242,96
290 DATA 160,3,142,242,63,189,229,63,32,183,63,174,242,63,232,136
300 DATA 208,240,96,141,243,63,140,245,63,142,244,63,32,210,255,174
310 DATA 244,63,172,245,63,173,243,63,96,169,13,32,183,63,96,32
320 DATA 228,255,201,32,240,1,96,32,228,255,201,32,208,249,96,0
330 DATA 0,73,78,84,70,76,79,83,84,82,0,3,6,0,0,0,0,0
```

5.7 CROSS (PRODUCES CROSS REFERENCE FOR BASIC COMMANDS

This auxiliary program produces a list of all BASIC Keywords and indicates in which lines they were used. In the lines from 63460 onwards, you can indicate the keywords which interest you and their codes (the Codes or Tokens of the BASIC Keywords are listed in chapter 6.1). If you want to add one of your own, type it in from 63480 together with the Token and increment the value in line 63450.

PROCEDURE

Type in the program and store it. Load the program which interests you and MERGE it with the CROSS Program (see 5.5 MERGE). Type: RUN 63000 and answer the question if output is to be on screen or printer. Be a little patient because the program has to go through the whole BASIC Program for each keyword.

```
63000 RESTORE 63450 : READ NU : DIM KW$(NU),K(NU)
63010 FOR I=1 TO NU : READ KW$(I),K(I) : NEXT I
63020 SCNCLR : INPUT "SCREEN OR PRINTER (S/P)";Q$
63030 IF Q$="P" THEN OPEN 4,4 : CMD 4 : GOTO 63060
63040 IF Q$<>"S" then 63020
63050 :
63060 FOR I=1 to NU
63070 : K=K(I)
63080 : PRINT KW$(I) : CN=0 : CO=0
63090 : PT=PEEK(43)+PEEK(44)*256
63100 : LN=0
63110 : DO WHILE LN<>63000
63120 :   NL=PEEK(PT)+PEEK(PT+1)*256
63130 :   LN=PEEK(PT+2)+PEEK(PT+3)*256
63140 :   IF LN=63000 THEN EXIT
63150 :   E=0
63160 :   J=PT+4
63170 :   DO WHILE J<>PT+80
63180 :     CH=PEEK(J)
63190 :     IF CH=34 THEN GOSUB 63320 : IF E THEN EXIT
63200 :     IF CH=K THEN GOSUB 63390
```

```
63210 :      IF CH=0 THEN EXIT
63220 :      j=j+1
63230 :    LOOP
63240 :    PT=NL
63250 : LOOP
63260 : IF CN<>0 THEN PRINT
63270 : PRINT "AMMOUNT=";CN : PRINT
63280 NEXT I
63290 IF Q$="D" THEN PRINT#4 : CLOSE 4
63300 END
63310 :
63320 DO WHILE CH<>0
63330 : J=J+1 : CH=PEEK(J)
63340 : IF CH=34 THEN EXIT
63350 : IF CH=0 THEN E=1 : EXIT
63360 LOOP
63370 RETURN
63380 :
63390 IF CO=6 THEN CO=0 : PRINT
63400 CO=CO+1 : LN$=STR$(ln) : LN$=MID$(LN$,2,LEN(LN$))
63410 IF LEN(LN$)<5 THEN LN$=MID$("00000",1,(5-LEN(LN$)))+LN$
63420 PRINT LN$;"   "; : CN=CN+1
63430 RETURN
63440 :
63450 DATA 10
63460 DATA DO,235,EXIT,237,FOR,129,GOSUB,141,GOTO,137
63465 DATA IF,139,LOOP,236,NEXT,130
63470 DATA RETURN,142,WHILE,253
```

6.1 THE TOKENS OF BASIC KEYWORDS

As you perhaps know, each BASIC command is stored in the program as so called token i.e. it appears in the memory of the C 16 as a byte and not as a whole word, which saves a lot of memory. Listed below are all the keywords and their tokens.

TOKEN (DEC.)	TOKEN (HEX.)	BASIC KEYWORD
128	$80	END
129	$81	FOR
130	$82	NEXT
131	$83	DATA
132	$84	INPUT#
133	$85	INPUT
134	$86	DIM
135	$87	READ
136	$88	LET
137	$89	GOTO
138	$8A	RUN
139	$8B	IF
140	$8C	RESTORE
141	$8D	GOSUB
142	$8E	RETURN
143	$8F	REM
144	$90	STOP
145	$91	ON
146	$92	WAIT
147	$93	LOAD
148	$94	SAVE
149	$95	VERIFY
150	$96	DEF
151	$97	POKE
152	$98	PRINT#

153	$99	PRINT
154	$9A	CONT
155	$9B	LIST
156	$9C	CLR
157	$9D	CMD
158	$9E	SYS
159	$9F	OPEN
160	$A0	CLOSE
161	$A1	GET
162	$A2	NEW
163	$A3	TAB(
164	$A4	TO
165	$A5	FN
166	$A6	SPC(
167	$A7	THEN
168	$A8	NOT
169	$A9	STEP
170	$AA	+
171	$AB	-
172	$AC	*
173	$AD	/
174	$AE	Δ
175	$AF	AND
176	$B0	OR
177	$B1	>
178	$B2	=
179	$B3	<
180	$B4	SGN
181	$B5	INT
182	$B6	ABS
183	$B7	USR
184	$B8	FRE
185	$B9	POS
186	$BA	SQR
187	$BB	RND
188	$BC	LOG
189	$BD	EXP
190	$BE	COS
191	$BF	SIN
192	$C0	TAN
193	$C1	ATN
194	$C2	PEEK

195	$C3	LEN
196	$C4	STR$
197	$C5	VAL
198	$C6	ASC
199	$CA	CHR$
200	$C8	LEFT$
201	$C9	RIGHT$
202	$CA	MID$
203	$CB	GO
204	$CC	RGR
205	$CD	RCLR
206	$CE	RLUM
207	$CF	JOY
208	$D0	RDOT
209	$D1	DEC
210	$D2	HEX$
211	$D3	ERR$
212	$D4	INSTR
213	$D5	ELSE
214	$D6	RESUME
215	$D7	TRAP
216	$D8	TRON
217	$D9	TROFF
218	$DA	SOUND
219	$DB	VOL
220	$DC	AUTO
221	$DD	PUDEF
222	$DE	GRAPHIC
223	$DF	PAINT
224	$E0	CHAR
225	$E1	BOX
226	$E2	CIRCLE
227	$E3	GSHAPE
228	$E4	SSHAPE
229	$E5	DRAW
230	$E6	LOCATE
231	$E7	COLOR
232	$E8	SCNCLR
233	$E9	SCALE
234	$EA	HELP
235	$EB	DO
236	$EC	LOOP

237	$ED	EXIT
238	$EE	DIRECTORY
239	$EF	DSAVE
240	$F0	DLOAD
241	$F1	HEADER
242	$F2	SCRATCH
243	$F3	COLLECT
244	$F4	COPY
245	$F5	RENAME
246	$F6	BACKUP
247	$F7	DELETE
248	$F8	RENUMBER
249	$F9	KEY
250	$FA	MONITOR
251	$FB	USING
252	$FC	UNTIL
253	$FD	WHILE
254	$FE	NOT USED
255	$FF	

The book contains all the essential information that you need to know when using your Plus 4 and C16. All the important aspects are explained in detail, the items already covered in the manual received with the computers have been omitted.

The graphics and machine language are specially stressed and demonstrated with example programmes.

Graphics

- Graphic possibilities of built in video chip TED.
- High-Res, multicolour and extended colour mode in machine code and basic.
- Programming the raster interrupt.

Sound

- Music with basic command.
- Sound programming of TED in machine code.
- Interrupt control of music.

Machine Code

- Introduction course.
- Tips and Tricks for Beginners.
- Commands of 7501 Micro Processor.
- Introduction for using KERNAL routines.

Useful Data

- Detailed memory map with exact description of each peak/poke.
- Large comparison chart of CBM64 and C16 for easy conversion.
- Utilities.

ISBN 2-717-00101-5

5 012717 001015

PRICE £7.95

ANCO SOFTWARE, 4 WEST GATE HOUSE, SPITAL STREET, DARTFORD, KENT DA1 2EH.
Telephone: 0322 92513/92518